TRABALHO DE GRADUAÇÃO

Proposta de Prova de Conceito de Rede para Internet das Coisas (IoT)

Jorge Guilherme Silva dos Santos

Priscilla Gameiro Rega

Brasília, Dezembro de 2016

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Proposta de Prova de Conceito de Rede para Internet das Coisas (IoT)

Jorge Guilherme Silva dos Santos

Priscilla Gameiro Rega

Trabalho de Graduação submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Georges Daniel Amvamve Nze, Dr., ENE/UnB _____
Orientador

Prof Ugo Dias, Dr., EnE/UnB _____
Examinador Externo

Diego Martins de Oliveira, Esp., IFB _____
Examinador Externo

Agradecimentos

O primeiro agradecimento tem que ir ao nosso orientador Prof. Georges, pelo apoio e pelos direcionamentos precisos que nos ajudaram a manter o foco do trabalho. O segundo agradecimento é feito ao prof. Ugo, que mesmo não sendo orientador do trabalho foi o nosso orientador envolvendo outros assuntos e, por isso, também assume grande importância no trabalho.
Na sequência, agradeço ao meu irmão mais velho, Henrique, que foi quem me inseriu no mundo da tecnologia, mesmo que o escopo na época fosse diferente. Agradeço aos demais familiares, especialmente meus pais, que foram os primeiros a tornar possível o ingresso na Universidade..
Por fim, um último agradecimento aos nossos colegas de curso, que por quase cinco anos nos aguentaram e foram peças fundamentais para facilitar a vida e descontrair.

Jorge Guilherme Silva dos Santos

Queria agradecer a minha amiga Maysa por ter nos ajudado na aquisição dos equipamentos e por ter me apoiado de diversas formas ao longo desse tempo. Ao nosso orientador Georges por ter nos guiado de forma clara e divertida nos mantendo focados nos nossos objetivos futuros. Ao prof. Ugo que nos guiou na parte de pesquisas acadêmicas e sempre forneceu conselhos valiosos tanto academicamente como para o futuro profissional.
Aos colegas de curso que sempre nos apoiaram e nos acompanharam nesses 5 anos de aprendizado e que mostraram que nem só de conhecimento acadêmico a Universidade é feita. E
Por fim, agradeço a minha família que sempre me apoiou nas minhas escolhas e tornou possível eu me encontrar nesse curso de graduação e ser feliz no caminho escolhido.

Priscilla Gameiro Rega

RESUMO

Este projeto tem como objetivo mostrar a implementação de uma rede IoT em ambiente real utilizando um hardware de baixo custo e softwares *opensource*. Ainda que existam implementações parecidas descritas em artigos, sendo algumas delas implementadas com tecnologias ou protocolos de características distintas, esse trabalho apresenta uma outra abordagem de implementação de redes IoT frente às soluções já existentes utilizando o mesmo conjunto básico de protocolos. O objetivo é que os passos descritos sejam reproduzíveis para que futuros trabalhos possam ser realizados utilizando essa rede como base para estudar os assuntos que não foram aqui abordados. Os resultados obtidos mostram o funcionamento da rede e possibilidade de combinação dos protocolos apresentados, provando seu funcionamento. As diversas melhorias possíveis, principalmente as que envolvem segurança, são sugeridas como trabalhos futuros ao final do trabalho.

ABSTRACT

This project aims to implement a real-world IoT network using low-cost hardware and open-source software. Even thought there are scientific papers describing similar implementations, most of them discuss different technologies or protocols. This project shows another IoT network implementation approach using the same basic protocol. Some of the topics weren't addressed in this project. The goal is to present a step-by-step implementation enabling other projects to use it as a base to study related subjects. The results show the running network and the combination of the presented protocols, proving how it works. The different improvements that can be made are presented at the end in the form of future works

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

6LowPAN	*IPv6 over Low Power Wireless Personal Area Networks*
AICCU	*Automatic IPv6 Connectivity Client Utility*
ARM	*Advanced RISC Machine*
CDIR	*Classless Inter-Domain Routing*
CoAP	*Constrained Application Protocol*
DNS	*Domain Name Service*
DHCP	*Dynamic Host Configuration Protocol*
DHCPv6	*Dynamic Host Configuration Protocol version 6*
EUI-64	*Extended Unique Identifier 64 bits*
GCC	*GNU Compiler Collection*
GPIO	*General Purpose Input/Output*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
ICMPv6	*Internet Control Message Protocol version 6*
IEEE	Institute of Electrical and Electronics Engineers
IoT	*Internet Of Things*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
JSON	*JavaScript Object Notation*
MAC	*Media Access Control*
MTU	*Maximun Transmition Unit*
NAT	*Network Address Translation*
NDP	*Neighbor Discovery Protocol*
NOOBS	*New Out Of Box Software*
RADVD	*Router Advertisement Daemon*
RFC	*Request for Comments*
SDHC	*Secure Digital High Capacity*
SLAAC	*Stateless Address Autoconfiguration*
SSH	*Secure Shell*
SPI	*Serial Peripheral Interface*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
VM	*Virtual Machine*
WPAN	*Wireless Personal Area Network*

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O conceito de IoT (Internet das Coisas) vem crescendo em popularidade nos últimos anos, com cada vez mais presença do assunto até mesmo nas mídias mais populares. A Internet das Coisas está cada vez mais presente em trabalhos de pesquisa da comunidade científica, abordando áreas diferentes dentro desse abrangente conceito. As primeiras publicações em IoT abordavam principalmente uma introdução ao assunto e descreviam as tecnologias mais prováveis e possíveis casos de uso [Atzori, Iera e Morabito 2010]. Mesmo com o assunto amadurecendo, artigos mais recentes continuam fazendo uma descrição geral das tendências em tecnologias, padrões e casos de uso [Tan e Koo 2014], [Al-Fuqaha et al. 2015].

Porém, entre os artigos, alguns já buscam fazer alguma implementação real dos conceitos em IoT. Entre eles, vale destacar [Al-Fuqaha et al. 2014], que implementou uma rede de IoT real na cidade de Padova (Itália) para monitorar níveis de poluição e iluminação pública da cidade. O objetivo era analisar dados ambientais e detectar de maneira inteligente locais da cidade que apresentavam problemas de iluminação pública. Nesse artigo, é mostrada uma arquitetura de rede diferente da que é usada normalmente na Internet, pois dizia utilizar uma pilha de protocolos que incluía os protocolos CoAP (*Constrained Application Protocol*) na camada de aplicação e 6LoWPAN (*IPv6 over Low Power Wireless Personal Area Networks*) com 802.15.4 nas camadas de rede e transporte.

Esse artigo é a principal fonte de motivação para o trabalho pois apesar de descreverem a arquitetura, não citaram nenhum método, ferramenta, *framework* ou aplicativo de importância para realização do ambiente na cidade de Padova. Por isso, o objetivo desse trabalho é implementar uma rede para IoT utilizando essa pilha de protocolos para dispositivos de baixo consumo e poder de processamento (chamados de *constrained*) descrevendo as ferramentas utilizadas e os detalhes de implementação para facilitar a reprodução do ambiente. Essa arquitetura não é idêntica à implementada na cidade de Padova, porém, terá alguns dos mesmos protocolos e funcionalidades. Assim, surge a prova de conceito: o trabalho irá mostrar o funcionamento de uma rede real, incluindo a troca de mensagens entre os diversos elementos da rede IoT, evidenciando os protocolos e também as ferramentas e procedimentos utilizados para alcançar o objetivo da proposta desse trabalho.

1.2 OBJETIVO

Como o conceito de IoT é muito amplo, envolvendo desde a transmissão em baixa potência (camada física) até o tratamento dos dados colhidos por objetos (ciência, mineração de dados e sistemas de informação), é importante delimitar claramente o escopo do trabalho: trata-se de um trabalho focado na rede e na visualização do funcionamento dos protocolos envolvidos.

O trabalho foi desenvolvido ao redor desse tema. Dado o objetivo citado na seção anterior, foi definida uma rede e depois elaborado o passo a passo das implementações necessárias para cada um dos protocolos a serem utilizados na rede IoT. Contudo, como uma rede em si não tem propósito se não houver uma aplicação para ela, foi construída uma pequena aplicação para fins de demonstração. Vale observar que a aplicação não é o foco do trabalho, ela apenas serviu como um meio de demonstrar o funcionamento da troca de mensagens entre objetos da rede IoT usando uma distribuição Linux Ubuntu e aplicações de código aberto.

Foi decidido que serão abordados os seguintes assuntos:

1.2.1 Objetivos específicos

- Implementar e configurar o ambiente de rede IPv4 e IPv6 com o auxílio do túnel 6in4;
- Configurar o protocolo CoAP na camada de aplicação, que por consequência inclui o UDP na camada de transporte;
- Implementação, configuração e testes de redes sem fio de baixo consumo e baixa taxa de transmissão com o IEEE 802.15.4;
- Implementar e configurar o protocolo 6LoWPAN como camada de adaptação entre IPv6 e WPAN.
- Implementação de uma aplicação simples para teste de todos os itens descritos anteriormente.

Assuntos que não fazem parte da delimitação do tema poderão ser citados como trabalhos futuros na seção 5.

1.3 ORGANIZAÇÃO DO TRABALHO

A partir dos próximos capítulos, o trabalho é dividido em quatro partes principais: referencial teórico, metodologia e implementação, resultados e conclusão.

A primeira parte aborda as tecnologias, protocolos, ferramentas e linguagens de programação utilizadas, sendo assim a fundamentação teórica. Essa primeira parte busca explicar o básico do funcionamento de tudo que será mencionado ou utilizado nas demais partes do trabalho. Analogamente ao que é feito em alguns livros da área, a fundamentação teórica segue uma abordagem *top-down*, em que tudo que se refere à camada de aplicação do modelo TCP/IP é apresentado primeiro, ficando os conceitos envolvendo as camadas de rede e enlace por último.

A parte de metodologia é um grande conjunto contendo diversos procedimentos operacionais que foi pensada para ser uma sequência de tutoriais que podem ser reproduzidos para que outros trabalhos possam utilizar a rede proposta como base para alcançar outros objetivos de pesquisa na área de IoT. Nessa seção também são apresentados os elementos principais da rede e como eles deveriam ser configurados para se comunicar.

Então, a parte de resultados é composta principalmente de imagens e suas explicações mostrando o sucesso da implementação e das configurações principais. Os resultados são a prova de que a rede funciona e que os elementos se comunicam utilizando a pilha de protocolos desejada.

Por fim, o último capítulo apresenta a conclusão do trabalho e sugestões de trabalhos futuros com base nos assuntos que estão fora do escopo desse trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 WEB E CAMADA DE APLICAÇÃO

2.1.1 HTML

O HTML ou *Hyper Text Markup Language* especificado na RFC 1866 é um formato de dados simples utilizado para criar documentos de hipertexto que são portáveis de uma plataforma para outra. [IETF 2016]. De forma mais simples, é uma linguagem de marcação utilizada para páginas web que é baseada em hipertextos, ou seja, conjuntos de elementos ligados por conexões. Esses elementos podem ser textos, imagens, hiperlinks, dentre outros.

Ao fazer a requisição de uma página web por meio do protocolo HTTP é enviado no cabeçalho do pacote de resposta o tipo do conteúdo do pacote. Com essa informação o navegador saberá interpretar a marcação da página. Se a marcação da página for do tipo HTML, o cabeçalho terá a seguinte linha

```
Content-type: text/html
```

O conteúdo de um documento HTML é marcado por meio de etiquetas em que cada uma irá sinalizar o início e o fim de um elemento e irá descrever qual tipo de conteúdo está naquela posição do documento. Um documento HTML básico possui no começo do documento a instrução *DOCTYPE* que irá informar a versão do HTML ao *browser*. Após essa declaração seguem as etiquetas. As etiquetas básicas são: *html*, *head*, *title*, *body* [W3School]. Cada elemento pode ainda ter um atributo que irá fornecer informações adicionais sobre um elemento. O navegador então interpreta cada uma das etiquetas e monta a página que é exibida ao usuário.

2.1.2 Javascript

Javascript é uma linguagem de script orientada a objetos que é executável em diversas plataformas. [Mozilla 2016]. É mais comum encontrar Javascript em execução em páginas Web, onde o servidor que hospeda a página distribui o código e o mesmo é executado pelos navegadores Web nos dispositivos dos usuários que acessam as páginas. Dessa forma, é possível utilizar Javascript para manipular dinamicamente os elementos das páginas.

Atualmente, é muito comum encontrar Javascript em execução em servidores em um ambiente chamado NodeJS [NodeJS 2016]. Nesse caso, o motor de processamento de códigos Javascript do navegador Google Chrome (chamado V8) é utilizado nos servidores para executar códigos Javascript. Nesse ambiente, o processamento é totalmente assíncrono, o que permite ao servidor atender milhares de conexões HTTP concomitantes em um única *thread* no sistema operacional. Esse modelo se contrapõe ao modelo tradicional em que uma *thread* de um servidor Web é criada para atender uma conexão. Dessa forma aplicações escritas com NodeJS apresentam menor consumo de recursos de *hardware* e escalam mais facilmente.

Sendo o Javascript a linguagem e o NodeJS o motor de execução no *backend* de uma aplicação, o *frontend* executando em um navegador Web pode ser tanto o Javascript puro quanto o Javascript extendido por alguma biblioteca (como a jQuery) ou desenvolvido com o auxílio de algum *framework*, como AngularJS (Google) ou ReactJS (Facebook), por exemplo.

Uma forma tradicional de desenvolver para Web é utilizar a pilha LAMP (Linux, Apache, MySQL e PHP). Quando se utiliza uma pilha totalmente Javascript executando no servidor atendendo requisições HTTP com ExpressJS, no navegador do usuário com AngularJS e com banco de dados MongoDB (que armazena documentos na notação de representação de objetos do Javascript), tem-se uma nova pilha com o nome de *MEAN* [MEAN 2016], que é uma sigla para *MongoDB, ExpressJS, AngularJS e NodeJS*.

2.1.3 HTTP

O HTTP ou *Hypertext Transfer Protocol* é um protocolo da camada de aplicação que é especificado na RFC 2616. A sua primeira versão HTTP/0.9 foi criada inicialmente para a transmissão pura de dados. A sua próxima versão HTTP/1.0 suportava mensagens no formato MIME (*Multipurpose Internet Mail Extensions*), como por exemplo, a linguagem de marcação HTML[IETF 2016]. Atualmente encontra-se na versão 1.1 e já existe uma proposta para o HTTP/2.0 [IETF 2016].

O HTTP/1.1 padroniza a comunicação entre aplicações por meio de mensagens de requisição e resposta. Um exemplo de aplicações conversando são *browsers* e servidores web. Um *browser* inicia a comunicação mandando uma requisição, por exemplo pedindo uma página web específica, para o servidor web. No cabeçalho dessa mensagem de requisição seguem o método de requisição, versão do protocolo HTTP utilizado e a URL destino. Abaixo tem-se o exemplo de um pacote de requisição HTTP com algumas informações extras que o cliente pode mandar para o servidor.

```
GET www.site.com/hello.html HTTP/1.1
User-Agent: Mozlila/4.0
Accept-Language: en-us
```

Ao receber essa requisição, o servidor web responde com uma mensagem resposta contendo no cabeçalho o status da resposta, o código do status, a versão do protocolo HTTP e, no corpo do pacote, a mensagem MIME. Abaixo tem-se o exemplo de um pacote de resposta HTTP com algumas informações extras que o servidor pode mandar para o cliente.

```
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Win32)
Content-Type: text/html

!DOCTYPE HTML
<html>
  <head>
    <title>Hello!</title>
  </head>
</html>
```

A aplicação cliente dispõe de métodos para fazer requisições ao servidor. Os mais utilizados são: `PUT`, `GET`, `POST`, `HEAD` e `DELETE`. O método PUT permite que o cliente modifique dados no servidor. O método GET permite que o cliente resgate informações do servidor. O método POST permite que o cliente insira informações no servidor. O método HEAD permite que o cliente resgate somente o cabeçalho do pacote não resgatando junto o corpo da mensagem (útil para verificação de cópias em cache, por exemplo). O método DELETE permite que o cliente delete informações do servidor.

Em resposta a essas requisições, o servidor dispões de códigos de respostas para informar ao cliente qual o resultado da sua requisição. Os códigos possuem prefixos fixos para identificar categorias de respostas. São elas:

- 1xx: Informação
- 2xx: Sucesso
- 3xx: Redirecionamento
- 4xx: Erro do cliente
- 5xx: Erro do servidor

2.1.4 JSON

O JSON é uma notação para representação de objetos Javascript. Todos os objetos na linguagem podem ter uma representação no formato JSON. Em aplicações Web, o JSON acaba sendo utilizado como uma forma de armazenamento e troca de dados em que a entidade que envia os dados codifica os mesmos nesse formato e a entidade que recebe é capaz de decodificar e trabalhar com os dados. Essa notação é muito utilizada para criar serviços que trocam informações pela Web e para trabalhar com APIs públicas que fornecem serviços diversos. Por exemplo: A API de distâncias do Google , que é capaz de fornecer o tempo de deslocamento entre dois locais, aceita retornar respostas no formato JSON, conforme o exemplo abaixo (apresentado para ilustrar o formato):

```
{
   "destination_addresses" : [ "Esplanada dos Ministerios - Brasilia, DF, Brasil" ],
   "origin_addresses" : [ "Asa Norte, Brasilia - DF, Brasil" ],
   "rows" : [
      {
         "elements" : [
            {
               "distance" : {
                  "text" : "5,1 km",
                  "value" : 5099
               },
               "duration" : {
                  "text" : "18 minutes",
                  "value" : 1096
               },
               "status" : "OK"
            }
         ]
      }
   ],
   "status" : "OK"
}
```

2.1.5 CoAP

O CoAP (*Constrained Application Protocol*) é um protocolo da camada de aplicação feito para ser usado por dispositivos com pouca capacidade de processamento e baixo consumo de energia. O objetivo desse protocolo é ser simples para ser executado nesses dispositivos sem perder a compatibilidade com protocolo HTTP e as APIs REST construídas sobre ele. Há algumas diferenças fundamentais entre CoAP e HTTP: O formato de mensagem do protocolo é binário, ao contrário do HTTP que é textual, e seu funcionamento se dá sobre o UDP na camada de transporte, e não sobre o TCP.

Essas diferenças em relação ao HTTP existem para permitir que o protocolo funcione com menor requisitos de desempenho. A adoção do UDP, por exemplo, é uma vantagem em relação ao uso do TCP devido à complexidade dos mecanismos de controle de congestionamento e o uso de buffers, que encarece o custo de hardware e dificulta a miniaturização de dispositivos. As características de baixo consumo de energia e baixa complexidade do CoAP o tornam atrativo para ser utilizado principalmente em aplicações M2M (*Machine to Machine*) na IoT (Internet das Coisas) [IETF 2014].

O protocolo é feito para funcionar sob uma abordagem *Request - Response*, em que requisições são feitas a um servidor, que retorna respostas de acordo com as requisições. Os métodos utilizados para fazer as requisições são praticamente os mesmos que são utilizados com o HTTP (`GET, POST, PUT, DELETE`) com a adição de um novo, chamado `OBSERVE`. A compatibilidade se extende às mensagens de resposta, que incluem, por exemplo, os códigos `200, 404, 500`. Isso foi feito para permitir a utilização de Proxies *HTTP - CoAP*, o que permitiria que dispositivos utilizando CoAP sejam compatíveis com os serviços atualmente disponibilizados em HTTP.

Apesar de funcionar com UDP, o CoAP é capaz de garantir confiabilidade nas mensagens trocadas por ele. Dois modos de funcionamento são possíveis: `NON`, que funciona sem obrigar confirmação das mensagens recebidas pelo receptor, e `CON`, que obriga que as mensagens recebidas sejam reconhecidas para que haja um mínimo de garantia de confiabilidade. A utilização do UDP permite que o protocolo funcione em um modo *assíncrono*, utilizando o método `OBSERVE`, que é especificado na RFC 7641 [IETF 2015].

O CoAP em si é especificado na RFC 7252 [IETF 2014] e não é ainda um padrão finalizado pelo IETF. Atualmente, se encontra na categoria de *Proposed Standard*, o que indica que sua padronização será concluída em breve.

2.2 CAMADA DE TRANSPORTE

2.2.1 TCP

O TCP ou *Transmission Control Protocol* definido na RFC 793 é um protocolo criado especificamente para oferecer uma comunicação confiável entre dois processos rodando em máquinas diferentes interligadas por um conjunto de redes [IETF 1981]. Considerando que esse conjunto de redes pode ser somente um salto ou um complexo com vários grafos, o TCP foi projetado para se adaptar dinamicamente, sendo um protocolo extremamente robusto diante dos vários pontos de falha que podem ocorrer.

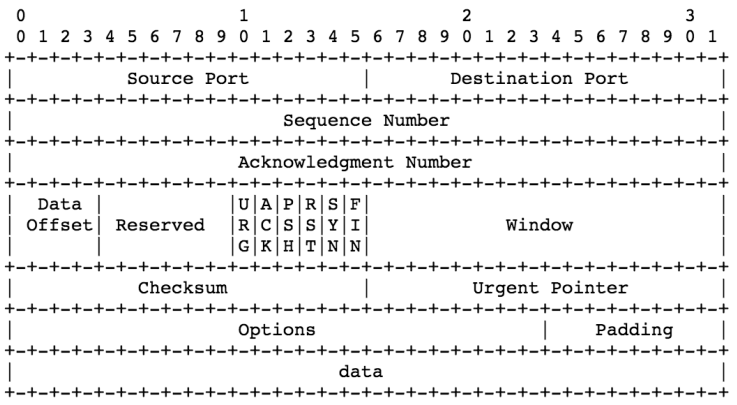


Figura 2.1: Cabeçalho do Pacote TCP [IETF 1981]

Um ponto importante é o serviço oferecido pela camada de rede para a camada de transporte. A camada de rede não oferece uma entrega confiável prometendo somente o melhor esforço para a entrega dos pacotes. Dessa forma, o TCP necessita de algumas ferramentas para se recuperar de pacotes que foram corrompidos, perdidos, duplicados ou entregues fora de ordem pela camada de rede.

Na Figura 2.1 encontra-se o cabeçalho do pacote TCP, que reflete a sua complexidade. Os primeiros campos compõem-se das portas de origem e destino da comunicação. Elas são necessárias para identificar quais processos nas máquinas estão se comunicando. Os campos *Sequence Number* e *Acknowledgment Number* são utilizados para garantir a entrega ordenada dos pacotes. O TCP endereça cada um dos segmentos com um número sequencial e aguarda as confirmações (ACKs) para transmitir os próximos segmentos ou retransmitir caso perceba perdas. Do lado da máquina de destino, os números de sequencias dos pacotes são utilizados para ordenar os pacotes corretamente e descartar aqueles que chegarem fora de ordem.

Outra ferramenta utilizada pelo TCP para garantir a entrega dos pacotes é o controle de congestionamento. Ao controlar o fluxo da injeção dos pacotes na rede (utilizando as mensagens de confirmação) o TCP evita o congestionamento na camada de rede.

2.2.2 UDP

O UDP ou *User Datagram Protocol* também é um protocolo utilizado para comunicação de dois processos que rodam em máquinas diferentes interligados por uma rede. Esse protocolo é padronizado na RFC 768 [IETF 1980]. Entretanto, diferente do protocolo TCP, o UDP não garante a entrega dos pacotes no destino e, caso os pacotes cheguem, ele não garante a ordem correta dos pacotes. Ele também não precisa do estabelecimento de uma conexão antes de começar a transmissão de dados.

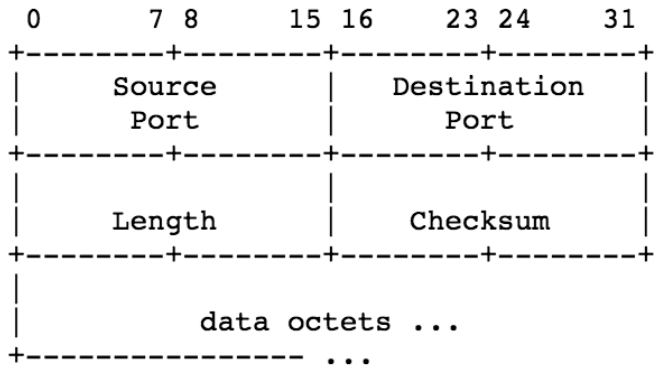


Figura 2.2: Cabeçalho do Pacote UDP [IETF 1980]

A figura 2.2 mostra o cabeçalho do pacote UDP. Para chegar ao seu processo destino, o UDP precisa somente das portas de origem e destino para identificar os processos que estão se comunicando. O campo *Length* possui o tamanho do pacote e o campo *Checksum* utilizado para checar a integridade do pacote na máquina destino. Comparado com o protocolo TCP, o protocolo UDP é consideravelmente mais simples.

Esse protocolo é utilizado por aplicações que não necessitam da entrega confiável dos pacotes ou que não se importa com a ordem em que os pacotes chegam. Aplicações como *streamming* de vídeos e VoIP aceitam uma taxa de perda de pacotes, não prejudicando a finalidade da aplicação. O protocolo também se encaixa para aplicações que trabalham com pacotes possuindo *payload* pequenos. Em caso de perda dessa pacote, após um intervalo de *timeout* envia-se novamente o pacote. Dessa forma não é necessário ter a configuração inicial da conexão que o protocolo TCP necessita nas suas comunicações confiáveis.

2.2.3 Firewall

Um firewall é um software criado para permitir o filtro de quais pacotes entram e saem pela rede. É utilizado dessa forma para proteção nas camadas de transporte e rede. Esse filtro é feito por meio de regras que são estabelecidas pelo administrador da rede. Os firewalls podem ser configurados diretamente nos *hosts* (os sistemas operacionais costumam incluir um firewall por padrão) ou podem também ser configurados para toda uma rede.

2.2.3.1 IPTables

O firewall instalado por padrão em várias distribuições Linux é baseado em um framework chamado *netfilter*, cuja associação mais comum é feita com o IPTables, que inclui as ferramentas para criação e gerenciamento dos filtros.

O IPTables é instalado por padrão em várias distribuições Linux. A sua configuração padrão em muitas distribuições é não filtrar nenhum pacote, ou seja, todo o tráfego passará por ele sem ser filtrado. Entretanto, as melhores práticas de segurança sugerem que existam filtros para bloquear a maioria do tráfego impedindo que pacotes indesejáveis trafeguem pela rede.

O IPTables possui diversas opções para filtrar os pacotes sendo os mais comuns baseados no protocolo de comunicação, nos endereços IP de destino e origem e nas portas dos processos. Ele possui três regras básicas: *Accept* para aceitar os pacotes, *Reject* para rejeitar o pacote e avisar a origem (usando ICMP) que aquele pacote foi rejeitado e *Drop* que irá rejeitar o pacote sem avisar a origem.

Essas regras são criadas e alocadas em uma das três cadeias de regras que filtram os pacotes: o *INPUT* que se refere aos pacotes de entrada, ou seja, que tem como destino final a máquina que recebe o pacote, o *FORWARD* que se refere a pacotes de passagem, ou seja, que não possuem origem nem destino na máquina que recebe o pacote e finalmente o *OUTPUT* que se refere a pacotes de saída, ou pacotes que tem origem na máquina para um outro destino qualquer.

Essa tabela é considerada a tabela de filtro e na sua configuração padrão tem os seguintes elementos:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Além dessa tabela o IPTables também oferece a tabela NAT, usada para criar conjuntos de regras que fazem algum tipo de tradução de endereço e/ou número de porta.

2.2.4 NAT

Com a escassez dos endereços IPv4 aumentando e a lenta migração para o IPv6, foi padronizado na RFC 3022 o NAT (*Network Address Translation*). O NAT consiste em realizar tradução de endereços, em que um endereço no cabeçalho de alguns datagramas IP é substituído por outro endereço. O SNAT (Source NAT) é uma das formas mais comuns, e sua ideia básica é atribuir somente um endereço IP para o tráfego na Internet (chamado de endereço público) enquanto na sua rede interna a empresa utilizaria um outro intervalo de endereços IP privados, ou seja, endereços que não poderiam se comunicar pela Internet. Assim, os endereços privados são substituídos pelo endereço público quando o datagrama é encaminhado da rede local para a Internet, conforme mostra a Figura 2.3.

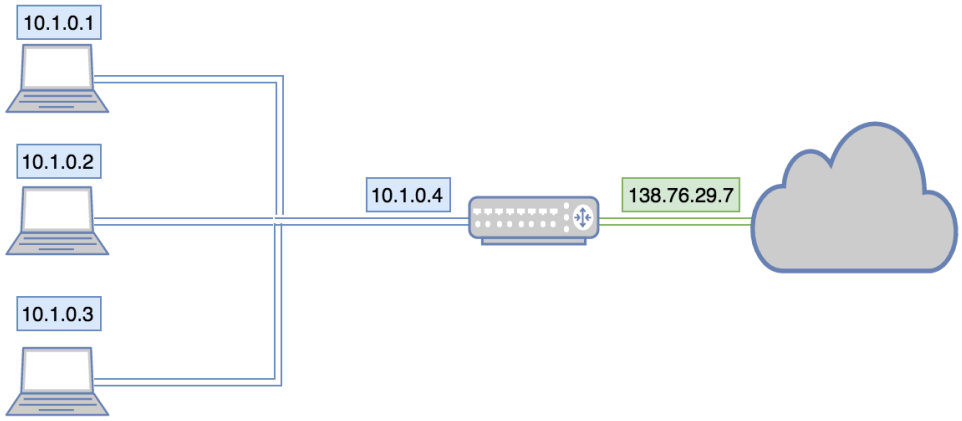


Figura 2.3: Conversão do endereço de origem pelo NAT. Adaptado de [Kurose e Ross 2012]

Para rede privada foram reservados três intervalos de endereços IP. Esses são endereços que obrigam o uso do SNAT para acesso à Internet:

```
10.0.0.0/8
172.16.0.0/20
192.168.0.0/16
```

2.3 CAMADA DE REDE

2.3.1 IPv4

O protocolo IP é padronizado pela RFC 791 e é o protocolo responsável pelo encaminhamento dos datagramas de dados (camada de rede) salto a salto na Internet. Para isso, ele utiliza endereços para identificar quaisquer *hgosts* na rede.

O endereço IPv4 é um número fixo de 32 bits divididos em 4 campos de 8 bits e é atribuído a interface de rede da máquina. Se a máquina possuir duas interfaces de rede, ela possuirá dois endereços IP. Ativos como roteadores por exemplo devem possuir mais de uma interface de rede possibilitando o roteamento dos datagramas para redes diferentes.

O endereço IP possui duas informações importantes: uma para identificar a rede a qual ele pertence e outra para identificar a máquina dentro da rede. Dependendo de como essa divisão das duas partes é feita, pode-se ter mais redes ou mais máquinas, de acordo com as necessidades de organização de cada rede. Originalmente essa divisão era fixa e feita por meio de classes, mostradas na Figura 2.4.

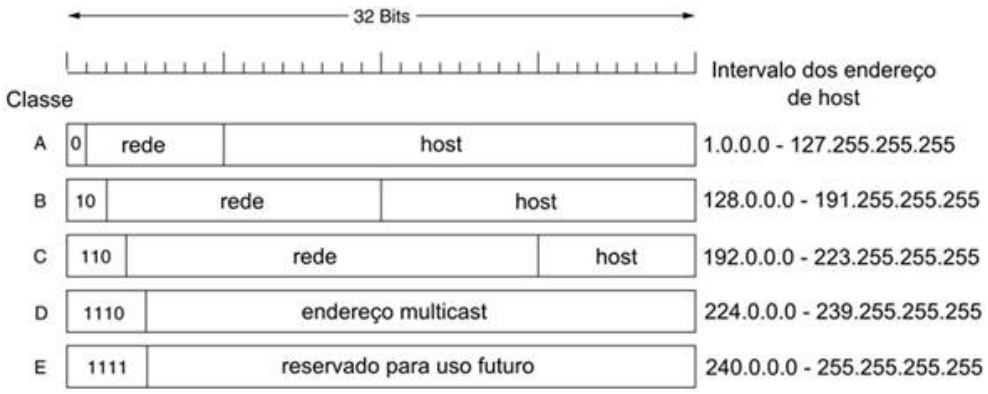


Figura 2.4: Tabela de classes de endereços IP.

Essa divisão em classes levou ao desperdício de endereços, o que contribuiu para a escassez de endereços IPv4. Dessa forma foi feita uma mudança na forma de endereçamento. Ao invés de seguir as classes, o endereço poderia ser subdividido de diversas formas, subdivisão identificada pelas máscas de rede de tamanho variável. A máscara de sub-rede indica qual parte do endereço IP está sendo utilizado para indicar a rede e qual parte está sendo utilizado para indicar a máquina. Comparando os dois modelos, o de classes possui máscaras de rede fixas enquanto o segundo modelo possui máscaras de rede variáveis.

2.3.2 IPv6

O IPv6 é uma nova versão do protocolo da Internet. Especificado na RFC 2460, foi projetado para ser o sucessor do IPv4 e destaca-se principalmente pela maior capacidade de endereçamento. Enquanto o endereçamento IPv4 previa o uso de endereços de 32 bits para identificar os *hosts* na Internet, o IPv6 prevê o uso de endereços de 128 bits. Além desse aumento, o IPv6 se destaca por simplificar o formato do cabeçalho e por acrescentar recursos importantes envolvendo privacidade e segurança [IETF 1998].

O IPv6 trouxe consigo mecanismos como o de descoberta automática de vizinhos (*Neighbor Discovery*) especificado na RFC 4861[IETF 2007]. Esse mecanismo permite que seja dispensado o uso de um servidor DHCP em um segmento de rede formado por *hosts* que utilizem IPv6, permitindo que os mesmos sejam capazes de determinar os endereços uns dos outros. Além disso, o protocolo traz a vantagem de dispensar a necessidade de compartilhamento de um IP público, como era feito no IPv4, uma vez que agora os endereços são suficientes para endereçar unicamente todos os *hosts* na Internet sem a necessidade de nenhum mecanismo para traduzir endereços válidos em endereços locais (como o NAT).

Entretanto, a adoção do novo protocolo é gradual apesar dos diversos esforços em acelerar a migração e da implacável redução na quantidade de endereços IPv4 disponíveis. Até Maio de 2016, o percentual de utilização de endereços IPv6 no Brasil segundo o CEPTRO BR[CEPTRO-BR 2016] não ultrapassou 12 % ao mesmo tempo em que o LACNIC[LACNIC 2016], órgão responsável pela alocação dos endereços IP para a região formada por América Latina e Caribe, informava haver apenas pouco mais de 1 milhão e 300 mil endereços IPv4 disponíveis.

Como uma mudança de protocolo nessa escala não é trivial, foram criados mecanismos de coexistência entre as duas versões dos protocolos, entre as quais estão a utilização de pilha dupla (*dual stack*) e a utilização de túneis 6in4 que encapsulam datagramas IPv6 dentro de datagramas IPv4, como ocorre com os *Tunnel Brokers*.

2.3.2.1 *IPv6 Tunnel Broker*

Trata-se também de um mecanismo de coexistência entre IPv4 e IPv6 na Internet durante a fase de transição. Um *Tunnel Broker* pode ser visto como um ISP IPv6 virtual, no sentido de prover conectividade IPv6 aos usuários já conectados à Internet IPv4. Basicamente, se baseia na implementação de serviços de túnel dedicados que são capazes de gerenciar automaticamente solicitações de túneis vindas dos usuários. A RFC 3053, que descreve o mecanismo, não padroniza um protocolo, apenas descreve o *framework* para criação dos túneis. Mais informações podem também ser encontradas na RFC 4013, que trata do túnel como um mecanismo de transição IPv6. [IETF 2001, IETF 2005].

Existem diversos serviços gratuitos de *Tunnel Broker* no mundo. O NIC.br (Núcleo Gestor da Internet no Brasil), através da sua iniciativa IPv6.br divulga instruções para utilização de IPv6 através de alguns desses serviços (*SixXS, Freenet6 e Hurricane Electric*) [NIC.br 2012]. Vale destacar o serviço oferecido gratuitamente pela SixXS, pois seus servidores (*Tunnel Servers* na nomenclatura da RFC 3053 e *Point of Presence* na nomenclatura da SixXS) estão espalhados por diversos pontos ao redor do mundo, inclusive no Brasil (Uberlândia)[SixXS 2016], o que minimiza os efeitos da latência extra introduzida pelo uso do túnel.

A Figura 2.5 resume o funcionamento dos túneis. O mecanismo todo pode ser dividido em três elementos: o nó do usuário, considerado um elemento pilha dupla na rede, o *tunnel broker*, que recebe os pedidos de configuração dos usuários, e os servidores de túnel (que também são os *tunnel endpoints*), que recebem as configurações do *broker* e utilizam as informações recebidas para criar e configurar os túneis. Assim, um usuário se conecta, registra e solicita ativação de túneis ao *Tunnel Broker*, que então gerencia toda a parte de criação de túneis em nome do usuário. Esse elemento também pode funcionar como balanceador de carga para distribuir o processamento e tráfego entre os servidores, distribuindo a criação dos túneis entre eles.

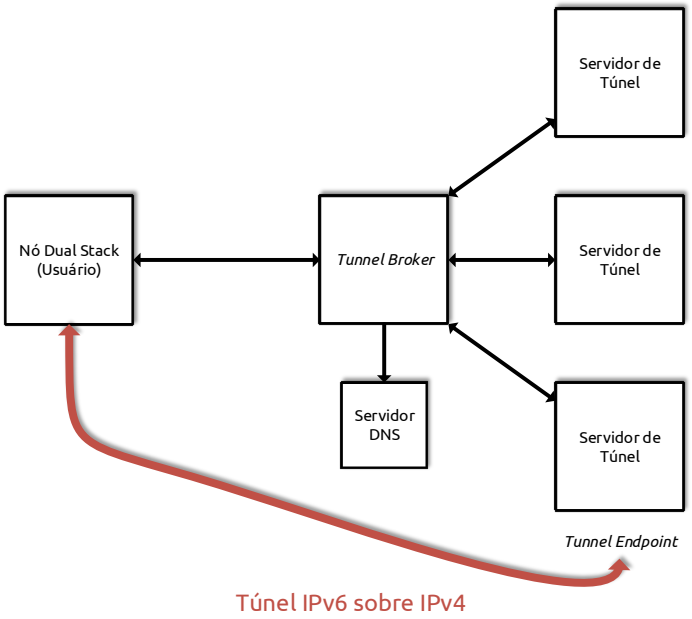


Figura 2.5: Diagrama representando o funcionamento do *Tunnel Broker*. Adaptado.[IETF 2001]

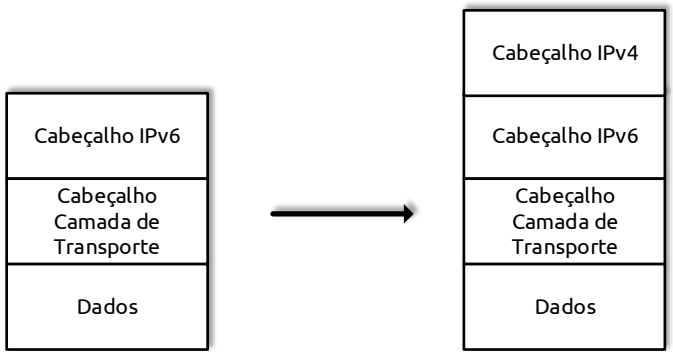


Figura 2.6: Encapsulamento de datagramas IPv6 em datagramas IPv4. Adaptado.[IETF 2005]

O nó do usuário pode ser tanto um *host* quanto um roteador, desde que seja capaz de funcionar no modo pilha dupla. O usuário precisa se autenticar com o túnel, que então deverá autorizar o usuário e, opcionalmente, efetuar cobranças (caso o serviço seja pago). Depois do procedimento e autenticação, o cliente precisa fornecer seu endereço IPv4, um nome a ser registrado no servidor DNS e, finalmente, a função (se o cliente é um *host* ou um roteador). Quando o *Tunnel Broker* recebe a solicitação do cliente, ele designa um servidor para atender o usuário, define um prefixo IPv6 global a ser alocado ao cliente, define um tempo de vida para o túnel, registra no seu servidor DNS o nome informado pelo cliente, configura o servidor de túnel e, finalmente notifica de volta as configurações relevantes ao cliente. Uma vez notificado, o cliente poderá se comunicar utilizando IPv6 através do encapsulamento dos datagramas IPv4 em datagramas IPv6, como mostra a Figura 2.6.

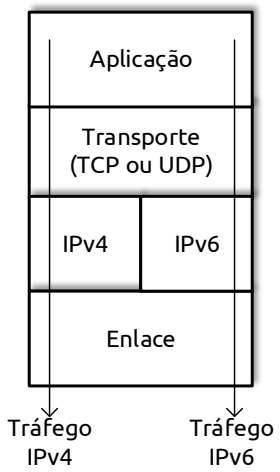


Figura 2.7: Funcionamento simplificado - Pilha Dupla. Adaptado.[IPv6BR 2012]

2.3.2.2 Dual Stack

O *dual stack* (ou pilha dupla) é uma das técnicas de convivência entre as redes IPv4 e IPv6 durante o período de transição entre os dois protocolos. Essa técnica consiste em manter em funcionamento duas pilhas de protocolo: uma com o IPv6 em funcionamento e outra com IPv4. Dessa forma, tráfego tradicional em IPv4 pode ser tratado mesmo que o servidor já esteja operando com IPv6.

Essa técnica é recomendada para o período de transição pois ela não favorece o uso de IPv4 (o que poderia atrasar a migração) e não afeta os serviços e seus usuários, que continuam a acessar tudo normalmente e de forma transparente. Para a Internet funcionando em IPv4, um *host* implementando pilha dupla se comporta como um *host* funcionando com IPv4. O mesmo ocorre com IPv6. Por isso, é necessário realizar duas configurações de rede em cada um dos nós que implementa pilha dupla: uma configuração para endereços IPv4 e outra configuração para endereços IPv6.

A Figura 2.7 mostra a coexistência das duas pilhas de protocolos em um único nó da rede. A Figura ressalta o fato das pilhas coexistirem simultaneamente e o nó responder tanto pelo tráfego IPv4 quanto pelo tráfego IPv6.

2.3.3 DHCP e DHCPv6

O DHCP (*Dynamic Host Configuration protocol*) especificado na RFC 2131 [IETF 1997] é um protocolo baseado no modelo *cliente - servidor* criado para passar informações de configuração para os *hosts* sem necessidade de intervenção manual. Além disso, possui um mecanismo de alocação automática de endereços para os *hosts* em uma rede, em que um *host* da rede solicita um endereço IP ao servidor DHCP que então escolhe um endereço entre os endereços que estão disponíveis, o reserva por um período limitado de tempo e responde à solicitação do cliente com o endereço escolhido. Além disso, sempre que o mesmo cliente repetir a solicitação, tenta entregar o mesmo endereço a ele. Esse endereço fornecido ao cliente pode ser completamente dinâmico ou pode ser definido estaticamente pelo administrador da rede.

Para a alocação de um endereço de rede, um cliente envia uma mensagem por *broadcast* para localizar os servidores DHCP, que respondem essa mensagem com uma mensagem do tipo `DHCPOFFER`, que inclui um endereço de rede disponível. Então o cliente envia uma mensagem `DHCPREQUEST` identificando o servidor DHCP esacolhido e o endereço que foi anteriormente ofertado. O servidor então pode atender a solicitação (enviando `DHCPACK` com os parâmetros de configuração) ou então negar (`DHCPNACK`) caso o servidor não seja capaz de atender por algum motivo.

O DHCPv6 especificado na RFC 3315 [IETF 1997] trata-se de um novo protocolo feito para funcionar com endereços IPv6. Ao contrário do DHCP para IPv4, os clientes se comunicam com os servidores DHCPv6 usando um endereço de *multicast*, podendo usar *unicast* em algumas circunstâncias. Além disso, trata-se de um protocolo de configuração *stateful*, em que os servidores gerenciam a atribuição dos endereços e salvam o estado de cada um dos clientes, permitindo uma administração de endereçamento centralizada. Apesar disso, ele deve apresentar compatibilidade com configuração automática de endereços *stateless* (SLAAC, RFC 4862 [IETF 2007]).

2.4 CAMADA DE ENLACE

2.4.1 Ethernet

O Ethernet é um padrão de transmissão especificado pelo grupo de trabalho 802.3. Da mesma forma que o protocolo IP utiliza os endereços IP para reconhecer as interfaces de rede das máquinas na rede, o Ethernet utiliza endereços MAC para reconhecer os adaptadores de rede das máquinas. Esse endereço é expresso em números hexadecimais em um conjunto de 6 bytes como por exemplo, 00:19:B9:FB:E2:58.

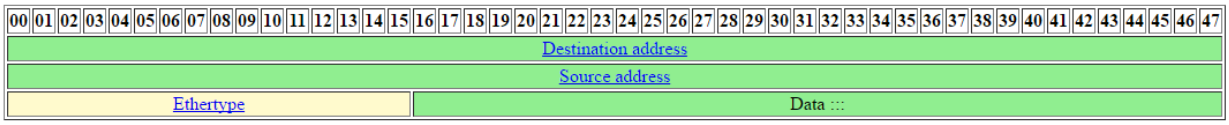


Figura 2.8: Cabeçalho do quadro Ethernet.Adaptado [IEEE]

Considerando uma rede local em que as máquinas possuem acesso a mesma infraestrutura cabeada, os endereços MAC são utilizados como endereços de origem e destino para a transmissão dos quadros. Na Figura 2.8 é mostrado o cabeçalho do quadro Ethernet com os endereços MAC para identificar as máquinas, um campo para identificar protocolo que foi encapsulado no quadro(por exemplo, IPv4) e finalmente os dados.

2.4.2 6LoWPAN

O 6LoWPAN se refere ao uso de datagramas IPv6 com redes pessoais (WPAN) que seguem o padrão IEEE 802.15.4. Esse padrão foi desenvolvido especialmente para dispositivos pessoais que são alimentados via bateria, com longos períodos de inatividade e possuem pouca capacidade de processamento e memória. É esperado também que aplicações que utilizem esse protocolo não estejam trabalhando com grande quantidade de dados. Por exemplo, sensores enviam pequenas informações em um intervalo de tempo consideravelmente longo, não necessitando de muitos bytes de *payload*.

O IPv6 é um endereço de 128 bits sendo maior que a versão IPv4 que possui o tamanho de 32 bits. Considerando que o tamanho esperado do MTU pelo IPv6 é 1280 bytes e que o quadro do padrão 802.15.4 tem MTU máxima de 127 bytes, é necessária uma etapa de adaptação para que o protocolo IPv6 seja devidamente encapsulado no quadro. Algumas ações como compressão o cabeçalho do IPv6 são tomadas para tornar prática a transmissão em 802.15.4. Se a compressão não puder ser implementada, será necessário realizar a fragmentação dos datagramas IPv6.

O cabeçalho padrão do IPv6 possui 40 bytes fixos que devem ser comprimidos para que o datagrama possa trafegar pelo enlace 802.15.4. Na Figura 2.9 é possível verificar como os 40 bytes fixos do cabeçalho são divididos. Com a compressão do cabeçalho é possível reduzir o cabeçalho para 2 bytes conforme mostrado na Figura 2.10. A compressão se utiliza de informações que são constantes dentro de uma rede local e que podem ser inferidas da camada de enlace. Dessa forma, o cabeçalho IPv6 não precisa carregar redundantemente essas informações na rede local, tornando possível a sua compressão. Esse projeto não irá trabalhar com compressão do cabeçalho, sendo necessário a fragmentação para que o datagrama possa ser transportado no padrão 802.15.4.

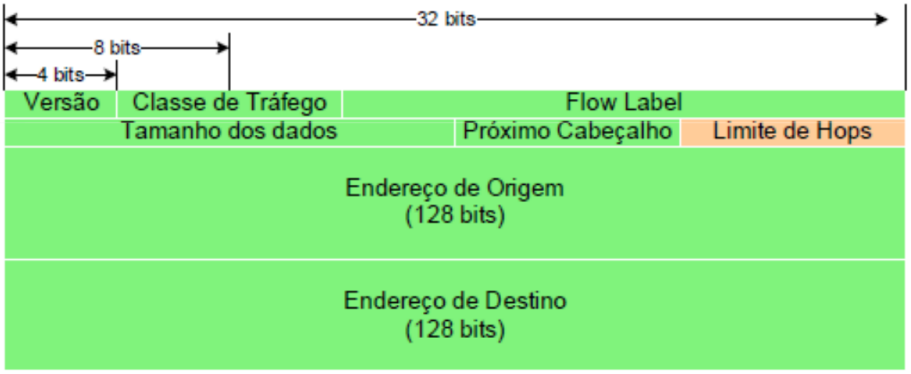


Figura 2.9: Cabeçalho do datagrama IPv6 padrão com tamanho fixo de 40 bytes.[NIC.br]

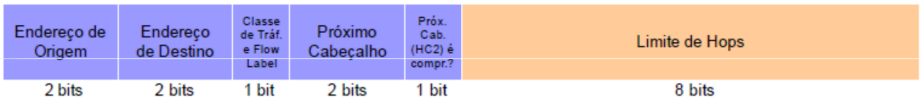


Figura 2.10: Cabeçalho do datagrama IPv6 comprimido com tamanho fixo de 2 bytes.[NIC.br]

2.4.3 Wi-Fi

O Wi-Fi é uma tecnologia que engloba vários protocolos de comunicação sem fios para redes locais (LANs) especificado pelo grupo de trabalho 802.11 do IEEE [IEEE 2016]. Trata-se de uma tecnologia da camada de enlace do modelo TCP/IP que utiliza o CSMA/CA como protocolo de acesso ao meio compartilhado.

O CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance* ou Múltiplo Acesso por Detecção de Portadora com Prevenção de Colisão) busca evitar a quantidade de colisões de transmissão, devido à complexidade (e consequentemente custos) de se realizar detecção de colisão, como é feito com o Ethernet, por exemplo. Seu funcionamento se baseia em escutar o canal antes de cada transmissão, para tentar detectar se há algum outro nó atualmente em transmissão na rede. [Kurose e Ross 2012]

O Wi-Fi normalmente utilizado nas LANs funciona no modo infraestrutura. Nesse modo, a arquitetura se baseia na presença de uma BSS (*Basic Service Set*) formada por um ponto de acesso sem fio (WAP - *Wireless Access Point*) e uma infraestrutura cabeada que fornece a função de roteamento à rede. Essa BSS possui um identificador (SSID), ou o "nome da rede", como e popularmente conhecido. Os elementos básicos de uma BSS são resumidos na Figura 2.11.

Atualmente a maioria das redes sem fio em empresas e residências utilizam a especificação do padrão IEEE 802.11. Por isso, a tecnologia está presente na maioria dos dispositivos à venda no mercado [Kurose e Ross 2012]. A última especificação do protocolo implementado para redes sem fio é o padrão IEEE 802.11ac, que permite taxas de transmissão teóricas na ordem de gigabits por segundo.

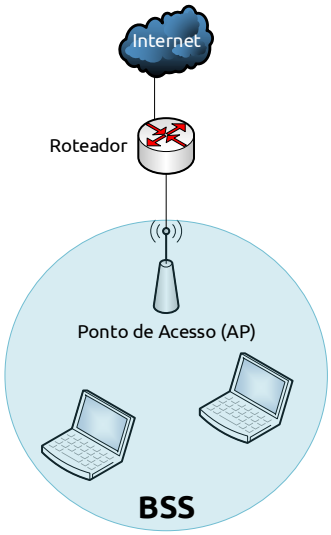


Figura 2.11: Elementos de uma BSS Wi-Fi Adaptado de [Kurose e Ross 2012]

2.5 HARDWARE

2.5.1 Raspberry Pi

O Raspberry Pi é um mini computador projetado para facilitar o aprendizado de programação de computadores. Atualmente na versão 3, ele possui componentes nativos suficientes para dar base para qualquer início de protótipo. Seguem as suas configurações de hardware:

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 1GB RAM
- 40 GPIO pins
- Full HDMI port e 4 USB ports
- Ethernet port e 802.11n Wireless LAN
- Bluetooth 4.1 e Bluetooth Low Energy (BLE)
- 3.5mm audio jack and composite video
- Camera interface (CSI) e Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

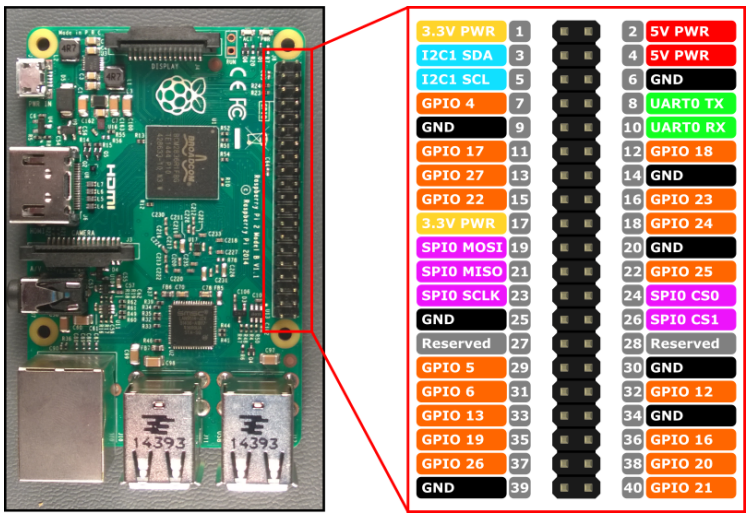


Figura 2.12: Mapeamento dos 40 pinos do Raspberry Pi 3 [Sunfounder].

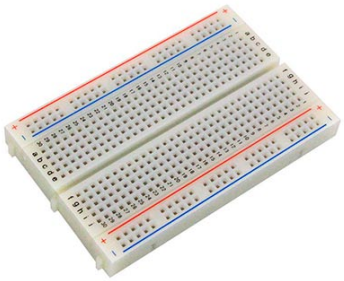


Figura 2.13: Protoboard para auxiliar montagem de circuitos [HUInfinito 2016].

Além do próprio Raspberry Pi algumas ferramentas são necessárias para auxiliar a montagem dos circuitos. O Raspberry Pi possui 40 pinos machos cujo mapeamento é ilustrado na Figura 2.12. Esses pinos podem ser estendidos para uma protoboard ilustrada na Figura 2.13 para facilitar a montagem dos sensores. Para realizar essa extensão são utilizados cabos macho-macho, macho-fêmea e fêmea-fêmea ilustrados nas Figuras 2.14, 2.15 e 2.16.

Para instalar um sistema operacional no Raspberry Pi 3 foi utilizado um cartão de memória Micro SD. Esse cartão pode ser adquirido com uma pré-instalação de um sistema operacional quando o microcontrolador é adquirido via um kit completo. Outra opção é fazer a instalação do sistema operacional diretamente no cartão. É indicado realizar a instalação do Raspian, uma linux modificado para trabalhar de forma mais eficiente com o Raspberry. Porém, nada impede que outras distribuições de Linux sejam instaladas, desde que tenham suporte ao hardware.

Uma vez que o sistema operacional está instalado, pode-se usar as portas USB para teclado e mouse e um monitor pode ser ligado na porta HDMI. A conexão com a Internet pode ser feita via a porta Ethernet ou via Wifi uma vez que o Raspberry Pi versão 3 possui Wifi nativo, não necessitando de um módulo extra como nas suas versões anteriores.

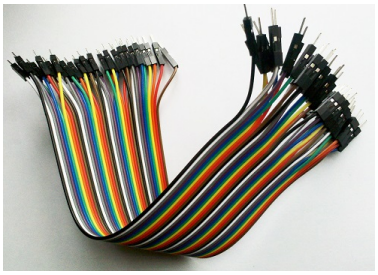


Figura 2.14: Cabos com duas entradas machos [HUInfinito 2016].

Figura 2.15: Cabos com uma entrada macho e uma entrada fêmea [HUInfinito 2016].

Para poder utilizar e monitorar os sensores com o Raspberry Pi 3 é necessário escrever códigos para interagir com os pinos e com os sensores. O Raspberry suporta várias linguagens de programação dentre elas a linguagem Python que será mencionada na seção 2.8.2.

A seguir, será feita uma breve descrição de alguns sensores utilizados com o Raspberry Pi:

2.5.1.1 DHT11 - Temperatura e Umidade

O DHT11 é um sensor de temperatura e umidade do ar. Ele utiliza três pinos para comunicação:

1. `SIG` - sinal;
2. `VCC` - Alimentação 5V;
3. `GND` - referência.

O pino SIG é usado pelo Raspberry Pi para enviar um sinal de inicialização (`START`) ao DHT11. Ao receber esse sinal, o sensor retorna uma palavra de 40 bits, dos quais:

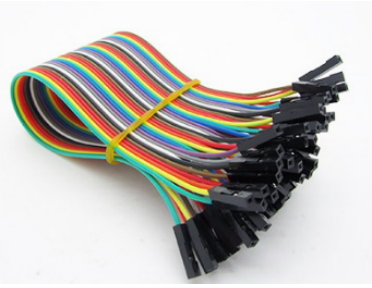


Figura 2.16: Cabos com duas entradas fêmeas [HUInfinito 2016].

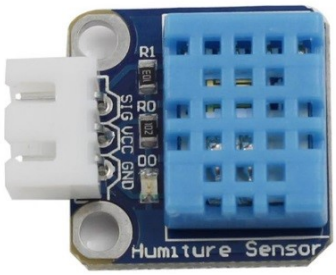


Figura 2.17: Sensor DHT11 [Sunfounder].

- 8 bits são a parte inteira da leitura de umidade;
- 8 bits são a parte decimal da leitura de umidade;
- 8 bits são a parte inteira da leitura de temperatura;
- 8 bits são a parte decimal da leitura de temperatura;
- 8 bits de checksum (para verificação de erros).

É necessário interpretar esses 40 bits para chegar ao resultado (em graus Celsius e umidade relativa). Existem bibliotecas prontas para trabalhar como DHT11 para diferentes plataformas, como o Arduíno e Raspberry Pi.

2.5.1.2 PCF8591 - Conversor Analógico-Digital

O sensor PCF8591 é utilizado para converter saídas analógicas em saídas digitais. Ele possui quatro entradas analógicas (AIN0, AIN1, AIN2 e AIN3), uma saída analógica (AOUT) e uma interface I2C (SCL e SDA), além das entradas de alimentação VCC e GND. O sensor pode ser visualizado na Figura 2.19.

Cada sensor em uma I2C é ativado enviando um endereço válido para o sensor. O endereço consiste em uma parte fixa e um parte programável como mostrado na Figura 2.18. A parte programável permite escolher em qual entrada analógica o sinal será convertido. Existem outras opções sendo o endereço válido 0x48 (formato hexadecimal) utilizado nesse projeto.

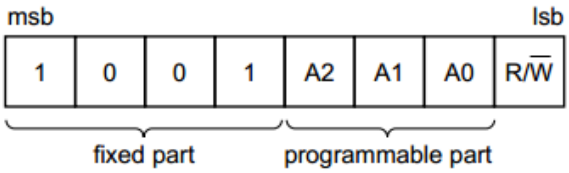


Figura 2.18: Endereço válido enviado do Raspberry Pi para o sensor PCF8591[Sunfounder].

Figura 2.19: PCF8591 - Conversor analógico-digital [Sunfounder].

2.5.1.3 Fotoresistor

O fotoresistor é um sensor que utiliza um resistor para indicar a intensidade da luz no ambiente. A medida que a intensidade da luz aumenta a resistência do sensor diminui mudando a tensão de saída do circuito. Esse valor de saída é analógico, dessa forma é necessário utilizar o sensor PCF8591 mencionado anteriormente para transformar a entrada analógica em uma saída digital. Na Figura 2.20 é ilustrado o sensor fotoresistor mostrando dois pinos machos para alimentação (VCC e GND) e um pino macho para saída dos dados analógicos (SIG).

2.6 SISTEMAS OPERACIONAIS

2.6.1 GNU/Linux

O Linux é um sistema operacional baseado no Unix que é disponibilizado sob a licença GPL (*Gnu Public License*) e, por isso, é usado por diversos desenvolvedores pelo mundo além de receber contribuições da comunidade, que engloba usuários, desenvolvedores e grandes empresas do mercado de tecnologia. O Linux em si é o núcleo do sistema operacional (*kernel*), que realiza as funções mais importantes e complexas de um sistema operacional, como acesso a *hardware* (permitindo sua abstração às aplicações), gerenciamento de recursos (CPU, memória, vídeo, etc), gerenciamento de usuários, escalonamento (ou agendamento) de processos, gerenciamento do sistema de arquivos, definição da estrutura de diretórios, etc

Figura 2.20: Sensor fotoresistor [Sunfounder].

Quando se combina o *kernel* Linux com um conjunto básico de aplicativos, drivers (módulos), bibliotecas, compiladores, etc, surge o nome *GNU/Linux*. O termo *GNU* se refere a esse conjunto básico de aplicativos que formam o projeto GNU (que inclui, por exemplo, o editor de texto Emacs e o compilador GCC, desenvolvidos pela *Free Software Foundation*). Por isso, o sistema operacional é definido como *GNU/Linux*, embora seja mais prático e comum utilizar somente o nome *Linux* [Morimoto 2009].

Existem versões de Linux pré-compiladas que acompanham um conjunto básico de aplicativos e, normalmente, um sistema de gerenciamento de pacotes para instalação de diferentes *softwares*. A cada uma dessas versões dá-se o nome de *distribuição* (popularmente chamada de *distro*). Existem centenas de distribuições Linux, entre as quais as mais utilizadas são atualmente o *Mint, Debian, Ubuntu, OpenSUSE e Fedora* [Distrowatch 2016].

2.6.1.1 Debian

O Debian é uma distribuição Linux que está entre as mais antigas e importantes. Totalmente livre (*opensource*) e desenvolvida pela comunidade, destaca-se pela sua estabilidade, uma consequência da política de revisão e congelamento de pacotes. Utiliza um sistema de empacotamento próprio (pacotes com extensão `.deb`) que pode ser utilizado em conjunto com repositórios de pacotes online com a ajuda do gerenciador de pacotes `apt-get`. Além de ser uma distribuição Linux, o Debian possui uma versão FreeBSD e uma versão com o kernel Hurd [Debian 2016].

2.6.1.2 Ubuntu

O Ubuntu é uma distribuição baseada na versão instável do Debian. Ao contrário do Debian, existe uma empresa por trás do Ubuntu, chamada *Canonical*. Por ser baseada no Debian, também utiliza pacotes `.deb` e o gerenciador de pacotes `apt-get`, mas conta com repositórios próprios contendo pacotes mantidos pela Canonical e pela comunidade. Sua versão *desktop* é voltada para usuários finais, mas a versão *Ubuntu Server* é uma das distribuições Linux para servidores mais utilizadas pelo mundo [Canonical 2016].

2.6.1.3 Raspbian

Versão do Debian GNU/Linux compilada para funcionar com o CPU do mini-PC *Raspberry Pi*, que é construído utilizano uma arquitetura ARM ao invés de uma arquitetura PC (x86). Trata-se do sistema operacional oficialmente suportado pela equipe criadora do *Raspberry Pi* e sua instalação já acompanha alguns softwares para desenvolvimento, como Python e o Java SE [RaspberryPi 2016].

2.7 OUTRAS LINGUAGENS DE PROGRAMAÇÃO

2.7.1 Java

O Java é uma linguagem de programação atualmente em desenvolvimento da Oracle [Oracle 2012]. Trata-se de uma linguagem fortemente baseada no paradigma de orientação a objetos. Além disso, é uma linguagem interpretada - ou seja, os códigos não são compilados para execução direta pelo sistema operacional - ao invés disso, os códigos são transformados em *bytecodes* e então passados para o interpretador, que é a JVM (máquina virtual Java).

A vantagem de uma linguagem interpertada que roda sobre uma máquina virtual própria é a portabilidade. Se um código é desenvolvido e fica funcional em uma JVM, ele pode ser portado para qualquer plataforma que execute a JVM. Isso permite que códigos em Java sejam executados até em plataforma de prototipação que executam um sistema Linux, supondo que esse sistema tenha instalado a JVM.

2.7.1.1 Framework Californium

O Californium é um framework feito em Java visando a implementação do protocolo CoAP. [Eclipse 2014] Ele foi desenvolvido pensando no cenário da Internet das Coisas sendo um framework para serviços *backend*. Além do Californium existem 4 outros projetos que abordam diferentes formas de utilizar esse framework. São eles: Actinium, Scandium, CoAP Tools e Connector. O framework pode ser encontrado no Github, possuindo repositórios com exemplos de aplicações e tutoriais de como instalar e utilizar o framework.

2.7.1.2 Servidor de Aplicação Actinium

O Actinium é um servidor de aplicação feito em Java baseado no framework Californium [Eclipse 2014]. O Actinium elimina a necessidade de criação do zero de um servidor de aplicação, uma vez que todas as funções principais já são implementadas e há uma organização prévia básica dos *resources* de cada objeto.

O Actinium permite que códigos sejam dinamicamente carregados em objetos através de mensagens baseadas no modelo REST utilizando o protocolo CoAP. Dessa forma, é possível inserir e atualizar códigos com uma mensagem PUT, obter o código e parâmetros de configuração com GET e atualizar parâmetros de configuração com POST. Isso permite que o comportamento do objeto seja alterado dinamicamente e remotamente, caso necessário.

Como exemplo, é possível imaginar uma situação hipotética em que um usuário deseja alterar o intervalo de envio de informações de um sensor ligado a um objeto da IoT. Se existir uma interface Web criada para facilitar essa interação, as requisições chegam a um servidor na central que então, utilizando CoAP, se comunica com os objetos alterando os parâmetros de configuração (ou até mesmo o código).

Apesar de escrito em Java, os códigos utilizados com o Actinium são feitos em Javascript. O interpretador Rhino para Java é utilizado para interpretar o código Javascript e executá-lo no servidor de aplicação. Por ser Javascript, sua integração é facilitada com ambientes desenvolvidos em Javascript (como interfaces executadas nos navegadores dos usuários).

Vale observar que nesse modelo os objetos de IoT podem ser os servidores de aplicação, permitindo que qualquer *host* autorizado (ou uma central com serviores especializados) possa se comunicar com os objetos desde que eles possuam endereços únicos e válidos na Internet, o que pode ser obtido com o uso de IPv6. Isso é especialmente útil quando se utiliza a função de *observe* do protocolo, em que basta que a central solicite uma relação de *observe* (que é uma mensagem GET especial [IETF 2015]) para que os objetos passem a enviar automaticamente quando necessário as informações para a central.

2.7.2 Python

O Python é outra linguagem focada no paradigma de orientação objetos. Em relação a outras linguagens, Python é considerada uma linguagem de mais alto nível pela forma como aborda seu tratamento com estruturas de dados. Todas essas características aliadas à sintaxe mais simples permitem o desenvolvimento rápido de certos tipos de aplicações e *scripts* [Python 2016].

Possui um grande repositório de pacotes e bibliotecas (PyPi), muitas das quais são desenvolvidas pela comunidade. Entre as bibliotecas, estão bibliotecas para desenvolvimento em plataformas de prototipagem (como o Raspberry Pi, por exemplo), permitindo criar dispositivos com simples funcionalidades de maneira muito rápida.

2.8 BANCOS DE DADOS

2.8.1 MongoDB

O MongoDB é um banco de dados *opensource* baseado em documentos. Todos os registros em um banco de dados MongoDB é um documento, que é uma estrutura de dados composta por diversos pares `campo:valor`. Os documentos são armazenados em uma coleção, sendo que um banco de dados pode ser composto por diversas dessas coleções [MongoDB 2016].

Cada documento armazenado em um banco MongoDB é muito semelhante a um objeto em representação JSON, dessa forma, é natural representar cada entrada no banco por um objeto em Javascript, tornando o desenvolvimento com MongoDB e Javascript (com o *runtime* NodeJS, por exemplo) mais natural para o desenvolvedor, pois os objetos em JSON são um formato nativo em Javascript.

Apesar da representação textual para o desenvolvedor, os dados são armazenados em formato binário (Binary JSON, ou BSON). Os objetos (documentos) inseridos em uma coleção não precisam necessariamente seguir o mesmo formato de dados. Objetos com novos formatos podem ser inseridos em uma mesma coleção.

2.9 VIRTUALIZAÇÃO

2.9.1 VMware - vSphere

O vSphere é uma solução de virtualização da VMware. Ele possui um hipervisor chamado de ESXi que é instalado em um servidor físico. O hipervisor irá abstrair os componentes físicos do servidor tornando possível a criação de máquinas virtuais que irão compartilhar esses recursos físicos. O hipervisor é transparente para a máquina virtual, ou seja, a máquina virtual acredita que tem total acesso aos recursos físicos. O vSphere é compatível com vários sistemas operacionais possibilitando a criação de máquinas Windows e Linux e suas distribuições. Além disso ele permite a criação de elementos de redes virtuais para segregar a comunicação entre as máquinas virtuais.

Dessa forma, possuindo apenas um servidor físico é possível criar um ambiente completo com várias máquinas virtuais e elementos de rede, facilitando por exemplo, a criação de ambientes de laboratório.

3 METODOLOGIA E IMPLEMENTAÇÃO

Este capítulo apresenta toda a metodologia e principalmente a implementação de todo o trabalho. Os procedimentos são descritos aqui para serem reproduzíveis de forma a possibilitar a implementação de sistemas parecidos. Vale observar, porém, que os códigos e configurações não serão mostrados na íntegra por questões de apresentação do texto.

3.1 COMPRA DOS DISPOSITIVOS

Foram adquiridos na loja virtual da *Amazon* americana dois kits para iniciantes que incluem o Raspberry Pi 3, cartão SD para instalação do sistema operacional, fonte de alimentação e uma caixa para proteção. Além disso, foi utilizado um terceiro Raspberry Pi 2 que já estava à disposição. Além desses componentes, foram adquiridos um conjunto de sensores diversos, dos quais dois foram utilizados: um fotoresistor (em conjunto com um conversor analógico para digital) e um sensor de temperatura e umidade (DHT11), que já é digital. Para montagem, foi adquirido um conjunto de cabos para ligação dos sensores à plataforma, popularmente conhecidos como *jumpers*.

Para mais informações, o sensor de temperatura e umidade foi descrito na seção 2.5.1.1, o fotoresistor e o conversor analógico para digital foram descritos nas seções 2.5.1.2 e 2.5.1.3, respectivamente. Os *jumpers* podem ser visualizados nas figuras 2.14, 2.15 e 2.16.

Por fim, dois módulos de rádio 6LoWPAN (IEEE 802.15.4) em 2,4 GHz foram adquiridos na loja do fabricante dos módulos: OpenLabs. Esses rádios permitem a utilização do protocolo 6LoWPAN (seção 2.4.2) pelos objetos. Os procedimentos de instalação serão descritos nas próximas seções.

3.2 MONTAGEM DOS OBJETOS

Foram realizadas duas montagens em paralelo em localizações geográficas diferentes. A primeira delas consistiu em utilizar apenas um Raspberry Pi funcionando como objeto e como *gateway* IPv6. Por limitações da quantidade de módulos de rádio, essa primeira rede não utiliza 6LoWPAN - toda a comunicação é feita utilizando Wi-Fi (IEEE 802.11n). Esse primeiro objeto possui o fotoresistor utilizado para medir a intensidade de luz no ambiente, cuja montagem é feita de acordo com o esquemático mostrado na Figura 3.1.

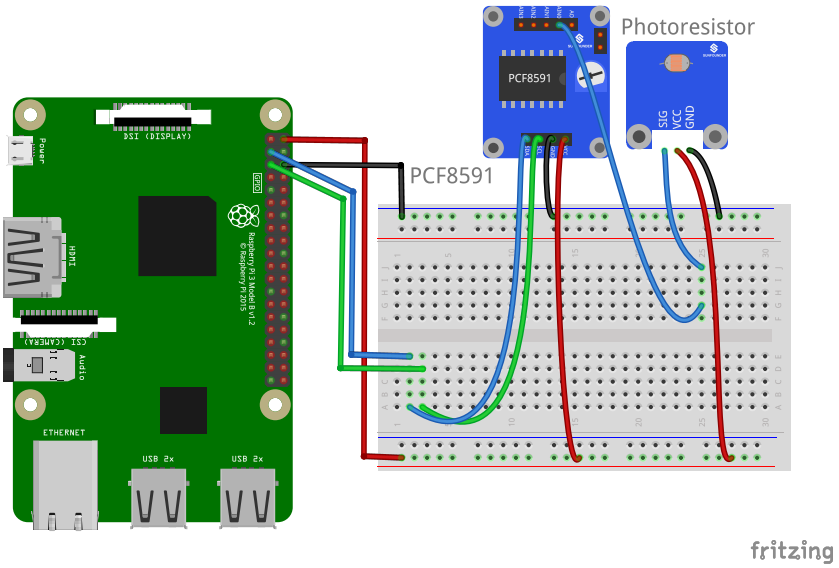


Figura 3.1: Esquema de montagem do fotoresistor com conversor analógico para digital. Adaptado de [Sunfounder]

A segunda montagem utiliza os módulos de rádio 6LoWPAN. Para isso, foram utilizados dois módulos de rádio 802.15.4, sendo um para cada Raspberry Pi (um que funciona como objeto e outro que funciona como *gateway* IPv6 e 6LoWPAN). O esquemático da Figura 3.2 mostra a montagem do objeto. A montagem do gateway não é mostrada pois é muito semelhante, basta retirar o sensor de temperatura.

Nessas figuras, é possível observar a ligação tanto dos módulos de rádio quanto do sensor de temperatura e umidade. Os rádios se comunicam com o Raspberry Pi utilizando SPI (*Serial Peripheral Interface*) e, por isso, devem ser ligados obrigatoriamente nos pinos mostrados, enquanto os sensores podem ser ligados em qualquer porta GPIO, desde que os ajustes no código sejam feitos apropriadamente.

3.3 CONFIGURAÇÃO DOS OBJETOS

Um dos dispositivos Raspberry Pi tem como objetivo ser o *gateway* IoT residencial. Isso significa dizer que todo o tráfego dos objetos presentes na casa será encaminhado por ele até a Internet. Além disso, um dos objetivos do trabalho é que a rede se aproxime da implementação de uma rede real, o que em IoT significa, entre outras coisas, utilizar IPv6 por dois motivos principais:

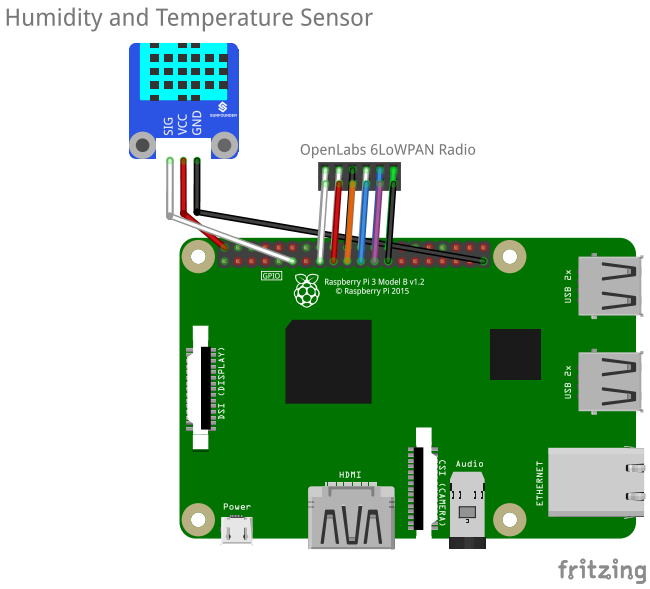


Figura 3.2: Esquema de montagem do sensor de temperatura e umidade DHT11 com módulo de rádio. Adaptado de [Sunfounder]

1. Necessidade de conexões externas de entrada - Cada objeto da rede executa um pequeno servidor de aplicação e, portanto, devem ser capazes de receber tráfego de entrada. Por mais que isso seja possível utilizando IPv4, existe a dificuldade inserida pelo NAT, que nesse cenário exigiria um redirecionamento de portas, que não é trivial de ser executado por usuários leigos e, por isso, se torna uma dificuldade para adoção de IoT em ambientes residenciais.

2. Necessidade dos objetos serem univocamente endereçáveis, o que com o crescimento da IoT se torna impossível se forem utilizados apenas endereços IPv4, que hoje obriga o uso de NAT.

Para possibilitar o uso de IPv6 com usuários residenciais, foi utilizado o serviço prestado pelo *Tunnel Broker* SixXS [SixXS 2016], fornecendo um túnel que permite conectar à Internet IPv6 uma residência que possui à disposição somente acesso IPv4. Isso é feito através do encapsulamento descrito na seção 2.3.2.1 e o serviço permite que usuários possuam simples túneis (com um endereço *endpoint*) ou uma subrede IPv6 /64 inteira.

3.3.1 Instalação do Sistema Operacional

Foi necessário primeiro instalar um sistema operacional no Raspberry Pi recém-adquirido. A distribuição recomendada pelo fabricante é o Raspbian [RaspberryPi 2016], baseado no Debian. Existem basicamente duas maneiras de instalar o Raspbian:

1. Utilizando um instalador automático, chamado NOOBS;

2. Gerar uma mídia de instalação (USB) e instalar o Raspbian diretamente.

Foi utilizada a segunda opção, o modo de instalação manual, pois foi necessário em um momento posterior trocar o Kernel do Sistema Operacional, como será mostrado na seção 3.3.2. Vale observar também que a instalação do sistema não incluiu ambiente desktop ou qualquer tipo de interface gráfica, para evitar que os gerenciadores de rede (como o NetworkManager) não influenciassem na configuração da rede.

Após a instalação do sistema, é preciso expandir o sistema de arquivos para que ele ocupe todo o espaço do cartão SD que contem o sistema. Para isso, é utilizado um utilitário simplificado de configuração que acompanha a instalação padrão do Raspbian:

```
# raspi-config
```

Nesse utilitário de configuração, bastou expandir o sistema de arquivos, escolher um *layout* e uma *timezone* adequados para a localização correta.

É também preciso atualizar os pacotes do sistema. Para isso, basta ligar o Raspberry a uma rede Ethernet cabeada. Se isso não for possível, é possível utilizar Wi-Fi, cuja configuração é definida no arquivo `/etc/wpa_supplicant/wpa_supplicant.conf`. Nesse arquivo, basta especificar o nome (SSID) da rede e sua respectiva senha:

```
network={
    ssid="minha-rede"
    psk="minha-senha"
}
```

Então a interface de rede sem fio (tipicamente `wlan0`) foi reabilitada:

```
# ifdown wlan0
# ifup wlan0
```

A atualização do sistema foi feita automaticamente utilizando o `apt`, o gerenciador de pacotes padrão da família de distribuições baseadas no Debian:

```
# apt-get update && apt-get upgrade
```

Após a atualização, foi preciso reinicializar o sistema, por conta de algumas atualizações de kernel e outros componentes básicos que costumam ser feitas.

3.3.2 Instalação do módulo de rádio WPAN (802.15.4)

O módulo de rádio para WPAN (802.15.4) utilizado no trabalho é montado pela OpenLabs e utiliza o chipset Atmel AT86RF233. Um par desses rádios foi utilizado: um para um objeto da rede e outro para o *gateway*.

No Raspberry Pi 3, esses módulos exigem recompilação do Kernel para incluir suporte ao chipset da Atmel e suporte geral à redes WPAN. Devido ao baixo desempenho do Raspberry para essas tarefas, foi utilizado o processo de compilação cruzada (*cross compiling*) que consiste em realizar a compilação em outro dispositivo com maior poder de processamento (no caso, um PC, que inclusive possui uma arquitetura diferente). Assim, os passos a seguir foram seguidos em um PC (arquitetura Intel x86_64 ou AMD64) com o sistema operacional Ubuntu 14.04 instalado.

3.3.2.1 Compilação do Kernel para WPAN

Primeiramente é preciso instalar algumas ferramentas e dependências:

```
# apt-get update
# apt-get install git make libncurses5-dev gcc-arm-linux-gnueabi
```

Dessas dependências, o `git` é um cliente em linha de comando que implementa o sistema de controle de versão de mesmo nome. Aqui ele é usado para clonar (fazer *download*) de repositórios do Github, uma forma bem popular de fazer a instalação de ferramentas *opensource*. O `make` é um utilitário que controla a geração de arquivos binários (normalmente compilação de código fonte) e é usado aqui para compilar e instalar o que precisar ser instalado pelo código-fonte. A biblioteca `libncurses5-dev` permite a criação de aplicações que exibam interfaces em linha de comando (útil quando não há servidor gráfico instalado), aqui usada para exibir a tela de configuração das opções do kernel. Por fim, o `gcc-arm-linux-gnueabi` é um dos compiladores usados para compilação cruzada do kernel Linux para arquitetura ARM, que inclui o caso do Raspberry Pi.

Assim como será feito mais adiante no próprio Raspberry, no PC utilizaremos um diretório exclusivo para manter os códigos que serão compilados no sistema: `/opt/src`. Esse diretório foi criado e o usuário padrão utilizado no sistema foi definido como dono da pasta para evitar que o processo de compilação seja executado como usário `root`:

```
# mkdir -p /opt/src/rpi_wpan
# chown <nome-do-usuario> /opt/src/rpi_wpan
$ cd /opt/src/rpi_wpan
```

Foi realizado o `clone` do repositório do *kernel*, dos arquivos de *firmware* e de algumas ferramentas de compilação, respectivamente:

```
$ git clone --depth 1 https://github.com/raspberrypi/linux.git \
    --branch rpi-4.7.y --single-branch linux-rpi2

$ git clone --depth 1 https://github.com/raspberrypi/firmware.git \
    --branch next --single-branch firmware

$ git clone --depth 1 https://github.com/raspberrypi/tools.git
```

O compildor GCC Linaro é utilizado. Isso é necessário para compilação do kernel para a CPU da família ARM Cortex do Raspberry Pi. É preciso exportar as variáveis de ambiente necessárias para que o compilador seja executado a partir das pastas onde foram baixadas as ferramentas acima:

```
$ export PATH=/opt/src/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64:/opt/src/
    tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/:$PATH
```

A configuração do novo kernel foi realizada para utilização do protocolo SPI com o transceiver 802.15.4 da Atmel:

```
$ cd /opt/src/raspi-wpan/linux-rpi2
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
```

Agora, foi necessário configurar o kernel para incluir suporte a 802.15.4 e 6LoWPAN:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

A opção `menuconfig` solicita a abertura de um menu de configuração do kernel, que abre uma interface baseada no Ncurses (vide a `libncurses` instalada anteriormente) no terminal para seleção das opções desejadas.

Essa tela permite habilitar o que é necessário (conforme mostrado abaixo). Alguns itens são marcados com a letra `M`, outros são marcados com `Enter`.

```
Networking support
  --> Networking options
    --> IEEE Std 802.15.4 Low-Rate Wireless Personal Area Networks support

Device Drivers
  --> Network device support
    --> IEEE 802.15.4 drivers

(foram marcadas todas as caixas, incluindo submenus)
```

Na próxima opção a ser modificada, foi inserida uma linha de configuração:

```
Boot options
  --> () Default kernel command string
```

Linha de configuração:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 root=/dev/mmcblk0p2 rootfstype=ext4 rootwait
```

Finalmente, foi iniciado o processo de *cross compile*:

```
$ CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm make zImage modules dtbs -j1
```

Esse comando define duas variáveis (`CROSS_COMPILE` e `ARCH`) necessárias para o processo de compilação (ou seja, são usados pelo script chamado pelo `make`). Então, o mesmo comando chama o `make` que recebe os parâmetros para compilação da imagem do Kernel, dos módulos de hardware e um terceiro parâmetro envolvendo a árvore de dispositivos do Raspberry Pi (DTB), uma vez que o objetivo é adicionar um novo dispositivo ao sistema. O último argumento desse comando indica o número de CPUs utilizados na compilação. No caso, o PC em questão possuí apenas uma CPU disponível, portanto foi utilizado o número 1. CPUs *quad-core* são comuns e nesses casos o número 4 poderia ter sido utilizado.

3.3.2.2 Instalação do novo Kernel recém-compilado

Com o aparelho desligado, o cartão de memória foi inserido no PC Linux utilizado para compilação. A organização utilizada previu o uso de pastas temporárias para montar os diretórios `boot` e `root` do cartão SD, diretórios que foram criados:

```
# mkdir -p /tmp/mnt/boot
# mkdir -p /tmp/mnt/root
```

Ao inserir o cartão de memória no sistema, qualquer um dos dispositivos `/dev/sdX` pode ser mapeado para o mesmo. É preciso verificar (utilizando, por exemplo, o comando `dmesg`) qual unidade alocada para o cartão de memória. Com isso identificado, foi realizada a montagem dos volumes, em que `sdX1` e `sdX2` foram substituídos pelos identificadores corretos dos volumes:

```
# mount /dev/sdX1 /tmp/mnt/boot
# sudo mount /dev/sdX2 /tmp/mnt/root
```

Os arquivos do novo kernel foram copiados para o cartão SD:

```
$ cd /opt/src/rpi_wpan/linux-rpi2
$ sudo cp arch/arm/boot/dts/*.dtb /tmp/mnt/boot
$ sudo cp arch/arm/boot/dts/overlays/*.dtb* /tmp/mnt/boot/overlays
$ sudo scripts/mkknlimg arch/arm/boot/zImage /tmp/mnt/boot/kernel7.img
```

Então, finalmente é feita a instalação. Esse comando utiliza o `sudo` com passagem da variável de ambiente `PATH`, o que permite que as bibliotecas corretas (especificadas anteriormente nessa variável de ambiente) sejam utilizadas:

```
sudo "PATH=$PATH" CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm \
  INSTALL_MOD_PATH=/tmp/mnt/root make modules_install
```

Por fim, foi feita a cópia dos arquivos de `firmware`:

```
$ cd /opt/src/rpi_wpan/firmware
$ sudo rm -rf /tmp/mnt/root/opt/vc
$ sudo cp -r hardfp/opt/* /tmp/mnt/root/opt
```

E, finalmente, as unidades foram desmontadas para o cartão ser levado de volta ao Raspberry, onde os procedimentos serão continuados:

```
$ sudo umount /tmp/mnt/root
$ sudo umount /tmp/mnt/boot
```

Com o sistema operacional instalado e o módulo de rádio configurado, foi realizada a configuração de rede, descrito a partir da próxima seção. Uma imagem do Raspbian com um novo (e modificado) Kernel foi gerada. Essa imagem pode ser copiada para ser aproveitada em diversos objetos da rede, evitando o demorado processo de compilação a cada nova instalação feita.

3.4 CONFIGURAÇÃO DE REDE DOS OBJETOS

3.4.1 Gateway IoT Residencial

3.4.1.1 Criação da Interface 6LoWPAN

Normalmente, em distribuições derivadas do Debian, a configuração das interfaces de rede podem ser feitas diretamente no arquivo `/etc/network/interfaces` ou através dos utilitários `ifconfig` e `ip address`. Por questões de compatibilidade, não foram utilizados esses scripts de inicialização, mas sim um script criado pela comunidade (RIOT-Makers [RIOTMakers 2016]) que permite habilitar e desabilitar a interface como se ela fosse um serviço do sistema operacional.

O diretório `/opt/src` foi escolhido para armazenar códigos-fonte de tudo que seria compilado manualmente. Esse diretório precisou ser criado e o usuário padrão do Raspberry (`pi`) foi configurado como dono dessa pasta (para que o processo de compilação e instalação não fosse feito como usuário `root`):

```
# mkdir /opt/src
# chown pi.root /opt/src
```

Então, o código-fonte foi baixado para essa pasta (através do clone de um repositório utilizando Git), da qual foram copiados os scripts para os locais corretos

```
$ git clone https://github.com/riot-makers/wpan-raspbian
$ cd wpan-raspbian
# cp -r usr/local/sbin/* /usr/local/sbin/.
# chmod +x /usr/local/sbin/*
```

Os scripts foram colocados em locais conhecidos (`/usr/local/sbin`) pois isso permite sua execução sem especificação de um caminho completo para eles - basta chamar o nome do script. Isso ocorre pois `/usr/local/sbin` é um dos diretórios definidos na variável de ambiente `PATH`, que define onde devem ser buscados os executáveis chamados pelo terminal.

Então, os scripts para integração com o `systemd` (que permitem tratar a interface como um serviço do sistema) foram copiados para os locais corretos:

```
# cp etc/default/lowpan /etc/default/.
# cp etc/systemd/system/lowpan.service /etc/systemd/system/.
```

O primeiro arquivo (`/etc/default/lowpan`) define configurações ou parâmetros para o serviço que será criado. O serviço em si é definido em `lowpan.service` no diretório onde são definidos os serviços no SystemD, que é `/etc/systemd/system`.

Com isso, é possível modificar o canal de operação do módulo de rádio 6LoWPAN no arquivo `/etc/default/lowpan`, bem como o endereço MAC da interface.

O serviço é então habilitado (para que possa inicializar automaticamente com a máquina):

```
# systemctl enable lowpan.service
```

Assim, a interface `lowpan0` é criada com o comando de inicialização de serviços:

```
# systemctl start lowpan.service
```

O comando `ifconfig lowpan` deverá mostrar a interface:

```
lowpan0   Link encap:UNSPEC  HWaddr 18-C0-FF-EE-1A-C0-FF-FF-00-00-00-00-00-00-00-00
          inet6 addr: fe80::1ac0:ffee:1ac0:ffff/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1280  Metric:1
          RX packets:18912 errors:0 dropped:0 overruns:0 frame:0
          TX packets:57349 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:2901827 (2.7 MiB)  TX bytes:5922183 (5.6 MiB)
```

3.4.1.2 Configuração do Túnel

Para permitir que o *gateway* e os objetos da rede se comunicassem utilizando IPv6, foi preciso configurar o mesmo como ponto de saída de tráfego para o túnel da SIXXS. Dessa forma, todo o tráfego de saída em IPv6 na direção da Internet IPv6 seria concentrado nesse elemento.

Os procedimentos oficiais para Ubuntu [SixXS 2015] foram utilizados para conexão com o Túnel. Os procedimentos para Ubuntu são compatíveis nesse caso, o que é facilitado graças ao fato do Raspbian ser baseado no Debian que, por sua vez, é também a base do Ubuntu. Os procedimentos seguidos são listados a seguir.

Primeiramente, foi preciso instalar a ferramenta *AICCU (Automatic IPv6 Connectivity Client Utility)* [SixXS 2015], que é a ferramenta recomendada pela SixXS para obter conectividade IPv6. Essa ferramenta funciona para usuários que desejam utilizar apenas um túnel ou uma subrede IPv6 inteira, além de permitir criação de túneis para usuários que se conectam através de *firewalls* e/ou NAT. Esses representam a maioria dos clientes de acesso à Internet residenciais, incluindo o cenário no qual a rede daqui foi montada.

O AICCU está disponível nos repositórios da versão do Debian utilizada, logo sua instalação foi feita com:

```
# apt-get install aiccu
```

O instalador possui um configurador automático. Ele solicita ao usuário, durante a instalação, as credenciais para autenticação na SIXXS e a identificação do túnel que será utilizado (pois cada usuário pode ter vários túneis). Isso é suficiente para o túnel funcionar e se ter conectividade IPv6 com o Raspberry.

Deve ser feita a verificação para testar se o AICCU foi iniciado corretamente:

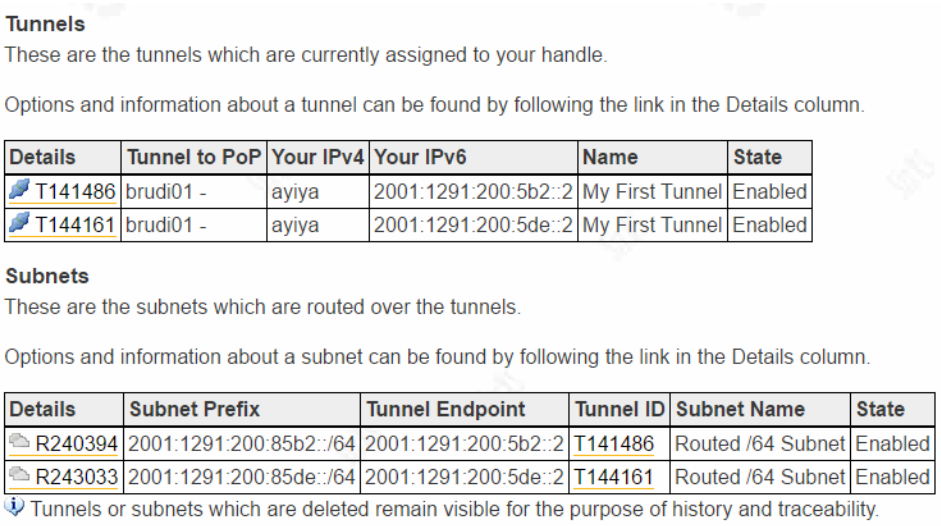
```
# service aiccu status
```

O instalador criou uma interface de rede virtual chamada `sixxs`, e isso também foi verificado. A saída retornada pelo comando `# ifconfig sixxs` é apresentada abaixo:

```
sixxs     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet6 addr: fe80::1091:200:5b2:2/64 Scope:Link
          inet6 addr: 2001:1291:200:5b2::2/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1280  Metric:1
          RX packets:636673 errors:0 dropped:0 overruns:0 frame:0
          TX packets:410429 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:673720842 (642.5 MiB)  TX bytes:42756503 (40.7 MiB)
```

Na saída do comando acima, foi verificado que o endereço IPv6 recebido identificado na segunda linha `inet6 addr` está de acordo com o informado na página da SixXs, conforme mostra a Figura 3.3. Essa Figura apresenta dois túneis que estão habilitados para uma conta com a SIXXS e duas subredes (*subnets*) que correspondem a esses dois túneis. Assim, os endereços apresentados são os que devem ser utilizados nas redes dos objetos.

Também há um outro endereço IPv6, o endereço do tipo *link-local*, que é utilizado para comunicações em um mesmo link e é configurado automaticamente de maneira não gerenciada (ou seja, sem uma entidade como um servidor DHCP, por exemplo). O outro endereço é válido na Internet e é do tipo *Global*.



Tunnels

These are the tunnels which are currently assigned to your handle.

Options and information about a tunnel can be found by following the link in the Details column.

Details	Tunnel to PoP	Your IPv4	Your IPv6	Name	State
T141486	brudi01 -	ayiya	2001:1291:200:5b2::2	My First Tunnel	Enabled
T144161	brudi01 -	ayiya	2001:1291:200:5de::2	My First Tunnel	Enabled

Subnets

These are the subnets which are routed over the tunnels.

Options and information about a subnet can be found by following the link in the Details column.

Details	Subnet Prefix	Tunnel Endpoint	Tunnel ID	Subnet Name	State
R240394	2001:1291:200:85b2::/64	2001:1291:200:5b2::2	T141486	Routed /64 Subnet	Enabled
R243033	2001:1291:200:85de::/64	2001:1291:200:5de::2	T144161	Routed /64 Subnet	Enabled

Tunnels or subnets which are deleted remain visible for the purpose of history and traceability.

Figura 3.3: Captura de tela do Painel de Controle do Usuário SixXS com identificação dos túneis utilizados [SixXS 2016]

Como o objetivo aqui foi utilizar o sistema como *gateway*, foi preciso configurar o *kernel* Linux para que ele realize encaminhamento de datagramas IPv6. Isso foi feito editando o arquivo `/etc/sysctl.conf` e trocando o valor da linha abaixo de 0 para 1:

```
net.ipv6.conf.all.forwarding=1
```

Na sequência, o comando abaixo foi utilizado para notificar o Kernel sobre o novo parâmetro para que ele efetivamente o aplicasse:

```
# sysctl -p
```

O próximo passo teria sido habilitar no *firewall* Linux (IPTables) o encaminhamento de pacotes. Porém, a instalação padrão do Raspbian vem com as regras completamente liberadas (política `ALLOW` por padrão na *chain* `FORWARD`), o que fez com que esse passo não fosse realmente necessário. Vale observar, porém, que em um ambiente de produção, é necessário tomar cuidado com as regras de *firewall*, especialmente nesse ambiente, em que os objetos possuem um endereço válido e podem ser diretamente acessados através de qualquer lugar da Internet.

3.4.1.3 Alocação automática de endereços

Para que a rede seja capaz de funcionar com SLAAC (*Stateless Address Autoconfiguration*), modo em que os *hosts* da rede configurem os endereços de suas interfaces automaticamente conforme a RFC 4862 [IETF 2007], foi preciso fazer com que o *gateway* enviasse *Router Advertisements* (parte do protocolo NDP - *Neighbor Discovery Protocol*) para dentro dessa rede, anunciando para a rede que ele é o roteador que deve ser usado como *default gateway*. Para isso, foi utilizado o *daemon* RADVD.

O *RADVD* em situações normais poderia ser instalado a partir dos repositórios oficiais do Debian utilizando as ferramentas incluídas de gerenciamento de pacotes. Contudo, foi tomada a decisão de utilizar a versão mais atual compilada a partir do código-fonte para evitar problemas com a rede LoWPAN [RIOTMakers 2016].

O RADVD utiliza o Flex, que deve ser instalado como dependência. O Flex, por sua vez, depende do *GNU Bison*. Portanto, sua instalação foi feita primeiro:

```
# apt install bison
```

Então, como usuário normal, o código fonte foi baixado para a pasta de códigos-fonte criada na seção 3.4.1.1.

```
$ cd /opt/src
$ wget https://sourceforge.net/projects/flex/files/flex-2.6.0.tar.bz2
$ tar xjf flex-2.6.0.tar.bz2
```

E foi feita a compilação do código:

```
$ cd flex-2.6.0
$ ./configure
$ make
```

A instalação foi feita como normalmente é feito por padrão em sistemas Linux:

```
# make install
```

Finalmente, o RADVD pôde ser instalado. Da mesma forma como foi feito anteriormente, *download* do código fonte e compilação:

```
$ cd /opt/src
$ git clone https://github.com/linux-wpan/radvd.git -b 6lowpan
$ cd radvd
$ ./autogen.sh
$ ./configure --prefix=/usr/local --sysconfdir=/etc --mandir=/usr/share/man
$ make
```

E, novamente, a instalação usual:

```
# make install
```

A configuração do RADVD é simples e foi feita com o uso da documentação como guia. O arquivo de configuração final é mostrado a seguir:

```
# Arquivo /etc/radvd.conf
interface lowpan0 {
    AdvSendAdvert on;
    AdvCurHopLimit 255;
    AdvSourceLLAddress on;
    MaxRtrAdvInterval 30;
    prefix 2001:1291:200:85b2::/64
            {
                    AdvOnLink off;
                    AdvAutonomous on;
                    AdvRouterAddr on;
            };
};
```

A configuração começa na segunda linha especificando em qual interface os anúncios serão feitos. No caso, eles serão feitos para os objetos que participam da rede WPAN, por isso os anúncios são feitos na interface `lowpan0` criada na seção 3.4.1.1.

O parâmetro `AdvSendAdvert on` indica que o roteador deve enviar *router advertisements* periodicamente e, também, responder a mensagens de solicitação provenientes da rede. O parâmetro `AdvCurHopLimit 255` é equivalente ao *Time to Live* da versão anterior do protocolo IP, ou seja, especifica o número de saltos que um pacote pode percorrer em uma rede antes de ser descartado. Nesse caso, as mensagens de anúncios de rotas estarão submetidas as desse limite. No contexto dessa rede de demonstração, esse valor não faz muita diferença, uma vez que poucos saltos existem na rede.

O parâmetro `AdvSourceLLAddress on` especifica que os RAs enviados por esse roteador devem especificar na mensagem o endereço de camada 2 (*link layer*) do roteador. Novamente, aqui a utilidade é demonstrar a funcionalidade - o resultado dessa configuração será mostrado na seção 4.3. O parâmetro `MaxRtrAdvInterval 30` faz com que o tempo máximo entre envios sucessivos de anúncios não solicitados seja de 30 segundos. Dessa forma, a cada 30 segundos o roteador envia um RA para a rede. Novamente, esse valor não necessariamente precisa ser tão baixo e o valor foi escolhido para demonstrar o funcionamento e facilitar os testes das configurações.

O parâmetro `prefix` especifica o prefixo que será anunciado pelo roteador. Ao receber o anúncio com esse prefixo, cada um dos objetos da rede será capaz de montar um endereço IPv6 válido completando o endereço com o identificador EUI-64 montado a partir do endereço MAC da interface de rede, conforme especifica a RFC 2373 [IETF 2016]. O prefixo CIDR `/64` especifica que são alocados 2^{64} endereços para a rede. Caso mais prefixos sejam necessários, deve-se anunciar mais prefixos $/64$ para evitar a divisão desse prefixo, conforme recomendam as boas práticas descritas na RFC 6177 [IETF 2016].

O parâmetro `AdvOnLink off` diz que os anúncios de roteador omitem informações sobre as propriedades *on-link* e *off-link* do prefixo, que são definidas na RFC 5942 [IETF 2010]. Essa opção existe pois, ao contrário do que ocorre com IPv4, os *hosts* que utilizam IPv6 e possuem um endereço (não *link-local*) configurado que pertence a uma determinada subrede não podem assumir por padrão que a subrede está diretamente conectada à essa interface. Assim, essa opção faz essa indicação de forma explícita [IETF 2010].

A opção `AdvAutonomous on` indica que um *host* da rede pode utilizar esse prefixo para configuração automática de endereços conforme especificado pela RFC 4862 [IETF 2007]. Por fim, a opção `AdvRouterAddr on` especifica que os anúncios devem enviar o endereço da interface de rede do roteador. Essa opção poderia estar desativada para forçar o roteador a enviar o prefixo de rede no lugar do endereço da interface de rede e também porque essa opção é especificada para *Mobile IPv6*.

Portanto, essa configuração faz com que o RADVD anuncie si próprio como *gateway* da rede especificada no parâmetro `prefix` (que é a mesma rede presente no painel da SixXS da Figura 3.3) a cada 30 segundos (parâmetro `MaxRtrAdvInterval`), Após isso, a configuração foi testada com:

```
# radvd -c
```

A saída do comando acima deverá retornar algo como o que é mostrado abaixo, indicando que a configuração foi feita corretamente.

```
[Oct 04 00:04:58] radvd (30813): config file, /etc/radvd.conf, syntax ok
```

Então, bastou iniciar o serviço para ativar essa configuração:

```
# radvd -d 1 -m syslog
```

O comando acima inicia o serviço com nível de detalhe dos logs definido como o nível mais baixo (nível 1), direcionando a saída de seus logs para o `syslog` do sistema (localizado em `/var/log/syslog`). É possível aumentar os detalhes que são enviados para o log (*debuglevel*) trocando o número 1 por um número mais alto, até o máximo de 4. Para visualizar em tempo real o funcionamento do RADVD, é possível monitorar o arquivo de log em tempo real:

```
# tail -f /var/log/syslog
```

Por fim, foi necessário configurar o serviço do RADVD para inicializar com o sistema. Uma das opções de fazer isso é utilizar o arquivo `/etc/rc.local`, que é um script chamado durante a inicialização do sistema operacional. Como todas as linhas presentes no arquivo são executadas durante a inicialização, basta incluir a linha desejada antes da linha contendo o comando `exit`, caso ela exista, conforme mostrado no exemplo abaixo:

```
# [...]

radvd -d 1 -m syslog

exit 0
```

Uma observação importante que deve ser feita a respeito do arquivo `rc.local` é que nos sistemas Linux que utilizam *systemd* ele não vem habilitado por padrão, é preciso habilitar o mesmo como um serviço. Para fazer isso:

```
$ sudo systemctl enable rc-local.service
```

3.4.1.4 Regras de Firewall

O *gateway* funciona também como *firewall* da rede pois todo o tráfego dos objetos deve passar por ele. Como na rede implementada os objetos não funcionam como clientes, mas sim como servidores, a preocupação se concentra nas regras de encaminhamento - ou seja, as regras da cadeia `FORWARD` do IPTables (*firewall* nativo dos sistemas Linux) - pois é nessa cadeia que são aplicadas as regras sobre o tráfego que é encaminhado aos outros dispositivos da rede (no caso, os objetos). A cadeia `INPUT` também é importante para restringir alguns tipos conexões de entrada realizadas diretamente com o *gateway* e também autorizar tráfego na situação em que o mesmo também se trata de um objeto na rede.

O conjunto de regras do IPTables relacionadas a IPv6 é gerenciada com o `ip6tables`, cujo funcionamento é análogo ao programa para IPv4 (`iptables`). Inicialmente, o comando `ip6tables -L` é usado para verificar as regras ativas. Por padrão, nenhuma regra está ativa e por isso a saída desse comando deve ser a mostrada abaixo.

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Caso existam regras indesejadas, todo o conjunto de regras pode ser limpado com o comando `ip6tables -F`.

Como as regras aplicadas diretamente usando o `ip6tables` são descartadas na inicialização subsequente do sistema, foram utilizados os *scripts* para salvar e recuperar as regras. Primeiro foi salvo o arquivo básico com o conjunto vazio de regras:

```
# ip6tables-save > /etc/network/ip6tables.rules
```

Esse arquivo foi editado para incluir as regras desejadas. Primeiramente, na cadeia `INPUT`, foi trocada a política padrão de `ACCEPT` para `DROP`, o que faz com que o *firewall* bloqueie toda a comunicação a menos das especificadas na lista correspondente. A mesma alteração foi feita para a cadeia `FORWARD`, deixando todo o trecho do arquivo de configuração conforme abaixo:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
```

As liberações foram adicionadas uma a uma, da regra mais geral para a regra mais restritiva. A primeira regra permite comunicação nas interfaces de *loopback* - que são usadas para comunicação de um objeto com ele mesmo:

```
ip6tables -A INPUT -i lo -j ACCEPT
```

A segunda regra trata de permitir ICMPv6 tanto na entrada quanto no encaminhamento. Por isso, foram adicionadas as linhas:

```
ip6tables -A INPUT -p icmpv6 -j ACCEPT
ip6tables -A FORWARD -p icmpv6 -j ACCEPT
```

Também é desejado permitir o acesso remoto via SSH ao *gateway*, para permitir sua gerência remota. Por segurança, esse acesso é autorizado somente à rede local. Com isso, foi adicionada a regra abaixo autorizando tráfego do protocolo TCP com porta de destino 22 (usada por padrão no servidor SSH) proveniente de qualquer *host* com endereço do tipo *link-local*:

```
ip6tables -A INPUT -s fe80::/10 -p tcp --dport 22 -j ACCEPT
```

A regra para os objetos libera tráfego UDP vindo de qualquer lugar desde que a porta de destino seja a porta em que o servidor de aplicação CoAP escuta (5683 por padrão):

```
ip6tables -A FORWARD -p udp --dport 5683 -j ACCEPT
```

Caso o próprio *gateway* seja utilizado também como objeto da rede, a regra acima deve ser inserida no encadeamento de regras de entrada (`INPUT`):

```
ip6tables -A INPUT -p udp --dport 5683 -j ACCEPT
```

Como nesse cenário os objetos são *hosts* executando sistemas Linux, é desejável que eles tenham acesso à Internet para instalação de pacotes e atualizações do sistema. Por isso, foi liberado tráfego HTTP, HTTPS e DNS de saída partindo da rede local (aqui referenciada por seu prefixo de endereços válidos). Vale observar também que quaisquer regras de saída necessárias para os *hosts* que se encontram na mesma rede dos objetos devem ser inseridas aqui.

```
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 80 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 443 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 53 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p udp --dport 53 -j ACCEPT
```

Uma última regra é necessária para as conexões que já foram previamente estabelecidas - seu tráfego deve ser autorizado:

```
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
```

O arquivo final é apresentado abaixo:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmpv6 -j ACCEPT
-A INPUT -s fe80::/10 -p tcp --dport 22 -j ACCEPT
-A INPUT -d ff00::/8 -j ACCEPT
-A INPUT -p udp --dport 547 -d ff00::/8 -j ACCEPT
-A INPUT -p udp --dport 546 -d ff00::/8 -j ACCEPT
-A INPUT -p udp --dport 5683 -j ACCEPT
-A FORWARD -p icmpv6 -j ACCEPT
-A FORWARD -p udp --dport 5683 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 80 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 443 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 53 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p udp --dport 53 -j ACCEPT
-A FORWARD -s 2001:1291:200:85b2::/64 -p tcp --dport 22 -j ACCEPT
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
```

As regras foram aplicadas com:

```
# ip6tables-restore < /etc/network/iptables.rules
```

Após aplicadas, as regras foram conferidas com `ip6tables -L`:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all      anywhere             anywhere
ACCEPT     ipv6-icmp    anywhere             anywhere
ACCEPT     tcp      fe80::/10            anywhere             tcp dpt:ssh
```

```
ACCEPT     all      anywhere             ff00::/8
ACCEPT     udp      anywhere             ff00::/8            udp dpt:dhcpv6-server
ACCEPT     udp      anywhere             ff00::/8            udp dpt:dhcpv6-client
ACCEPT     udp      anywhere             anywhere            udp dpt:5683

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     ipv6-icmp    anywhere             anywhere
ACCEPT     udp      anywhere             anywhere            udp dpt:5683
ACCEPT     tcp      2001:1291:200:85b2::/64  anywhere            tcp dpt:http
ACCEPT     tcp      2001:1291:200:85b2::/64  anywhere            tcp dpt:https
ACCEPT     tcp      2001:1291:200:85b2::/64  anywhere            tcp dpt:domain
ACCEPT     udp      2001:1291:200:85b2::/64  anywhere            udp dpt:domain
ACCEPT     tcp      2001:1291:200:85b2::/64  anywhere            tcp dpt:ssh
ACCEPT     all      anywhere             anywhere            state RELATED,ESTABLISHED

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Para fazer com que as regras sejam aplicadas durante a inicialização, uma das opções é carregar o script após a inicialização dos serviços da máquina utilizando o script em `/etc/rc.local`. Basta adicionar o comando ao arquivo antes da linha que contem `exit`, caso ela exista.

3.4.1.5 Detalhes de roteamento adicionais

Pode ter sido notado que não foi feita a configuração do endereço da interface interna `lowpan0`. Isso faz com que o roteador não conheça uma rota para alcançar a rede IPv6 interna (na qual estão os objetos). É possível resolver isso de duas formas diferentes: primeiro, é possível simplesmente adicionar uma rota estática para o prefixo de rede utilizado; a segunda opção é definir um endereço de IP fixo na interface, o que faz com que a rota seja adicionada automaticamente.

Para a segunda opção, basta executar, por exemplo:

```
$ ip addr add 2001:1291:200:85b2:0000:ffee:1ac0:ffff/64 dev lowpan0
```

O prefixo de rede pode, por simplicidade, ser o mesmo prefixo de endereços válidos anunciado pelo RADVD. O identificador EUI-64 pode ser copiado do endereço *link-local* da interface, conforme ilustra a saída do comando `ifconfig lowpan0` abaixo (que também mostra a configuração do endereço recém-adicionado):

```
lowpan0   Link encap:UNSPEC  HWaddr 18-C0-FF-EE-1A-C0-FF-FF-00-00-00-00-00-00-00-00
          inet6 addr: 2001:1291:200:85b2:0:ffee:1ac0:ffff/64 Scope:Global
          inet6 addr: fe80::1ac0:ffee:1ac0:ffff/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1280  Metric:1
          RX packets:18884 errors:0 dropped:0 overruns:0 frame:0
          TX packets:57270 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:2897645 (2.7 MiB)  TX bytes:5914381 (5.6 MiB)
```

O comando acima faz com que a configuração seja válida até a próxima reinicialização. Para tornar essa configuração permanente, foi utilizado o mesmo script de inicialização utilizado pelo RADVD e pelo IPTables. Assim, o final do arquivo `/etc/rc.local` é apresentado abaixo:

```
# [...]

radvd -d 1 -m syslog
ip addr add 2001:1291:200:85b2:0000:ffee:1ac0:ffff/64 dev lowpan0
ip6tables-restore < /etc/network/iptables.rules

exit 0
```

Com isso, a configuração do *gateway* está completa e, portanto, ele é capaz de rotear os dados dos *hosts* (ou objetos, nesse caso) para a Internet IPv6 através de um túnel e também permite a configuração automática dos mesmos, dispensando que configurações de rede adicionais tenham que ser feitas.

3.4.2 Demais Objetos da IoT

Uma vez que o cenário foi montado para distribuir endereços IPv6 automaticamente via RADVD, a configuração dos objetos da IoT acabam sendo extremamente simplificada. Assim, basta replicar a configuração da interface 6LoWPAN descrita na seção 3.4.1.1. Ao habilitar a interface de rede, um endereço IPv6 será atribuído automaticamente com SLAAC. Detalhes e provas do funcionamento serão apresentadas na seção 4.3.

3.5 CONFIGURAÇÃO DE REDE DA CENTRAL

A central de comunicação consiste em um conjunto de três servidores virtualizados (máquinas virtuais, ou VMs) hospedados em uma mesmo *host* físico em um datacenter na França. O *host* executa uma versão de avaliação (gratuita) do VMWare ESXi, enquanto todas as máquinas virtuais executam o Ubuntu 14.04 LTS.

Esse tipo de instalação foi preferido pois isso se aproxima de um ambiente real, em que máquinas virtuais em serviços de hospedagem são utilizados para executar o código das aplicações e fornecer os serviços aos usuários. Além disso, isso foi necessário para se obter conectividade IPv6 nos servidores, uma vez que um dos objetivos do trabalho é a comunicação direta da central com os objetos e a demonstração de funcionamento da pilha de protocolos para IoT, que consiste no uso de IPv6.

Os servidores funcionam com IPv4 e IPv6 simultâneamente (*dual stack*). Assim, a configuração de rede da central de comunicação pode ser dividida em três partes principais:

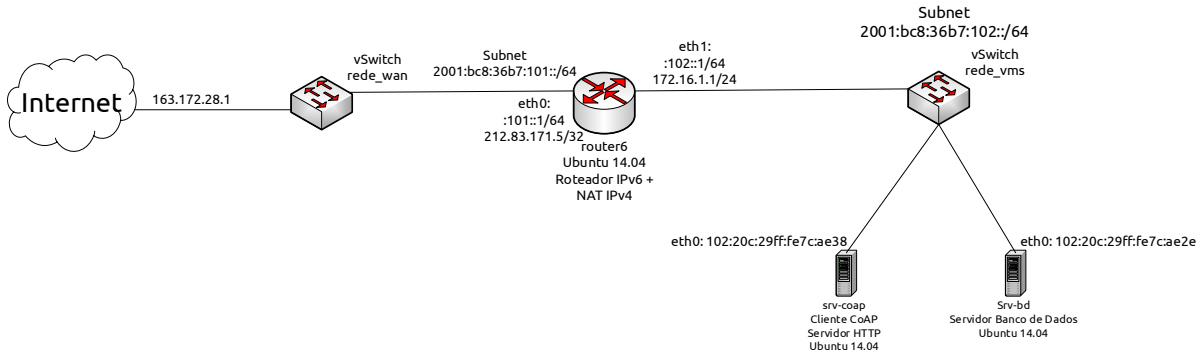


Figura 3.4: Diagrama mostrando a arquitetura de rede dos servidores [autores]

1. Configuração dos *switches* virtuais na plataforma de virtualização;
2. Configuração IPv4;
3. Configuração IPv6

A primeira parte consiste da configuração da rede na plataforma de virtualização (VMWare ESXi). Essa configuração é feita de acordo com a arquitetura de rede escolhida para o ambiente, conforme apresentado na Figura 3.4.

Nessa arquitetura, é possível identificar dois segmentos de rede, representados pelas interfaces de rede ligadas à máquina virtual que funciona como roteador. Cada uma dessas interfaces é ligada a um dos *switches*, que são virtuais. Um segmento de rede é o que dá acesso externo aos servidores - por isso é chamado de `rede_wan`. O outro segmento é o interno, que é o utilizando para comunicação entre as máquinas - por isso é chamada de `rede_vms`.

À esquerda do diagrama é possível identificar a saída de acesso à Internet através do switch `rede_wan`. Nesse *switch* está ligada a interface de saída do *host* virtualizador, que está ligada à rede da empresa que forneceu o serviço de hospedagem. Internamente, a interface interna (que é virtual) liga a máquina virtual chamada de `router6`. Essa máquina recebe esse nome pois ela é o roteador IPv6 da rede - ou seja, ela faz o roteamento do tráfego entre as duas redes: a rede externa (que possui somente um endereço alocado, a da própria interface externa da VM) e a rede das VMs. Portanto, essa máquina virtual funciona como *gateway* para o tráfego IPv6 de saída.

Além de roteador IPv6, a máquina virtual `router6` também faz roteamento IPv4. Porém, devido à disponibilidade de somente um endereço IPv4 válido (decorrente do custo de uso de cada um dos endereços), essa máquina utiliza NAT para que os servidores na `rede_vms` possam utilizar endereços IPv4 não válidos para roteamento na Internet. No caso, foi adotada a subrede `172.16.1.0/24`. Assim, além de se comunicar com IPv6 utilizando endereços de escopo global, as máquinas virtuais podem se comunicar utilizando IPv4 com endereços não roteáveis na Internet.

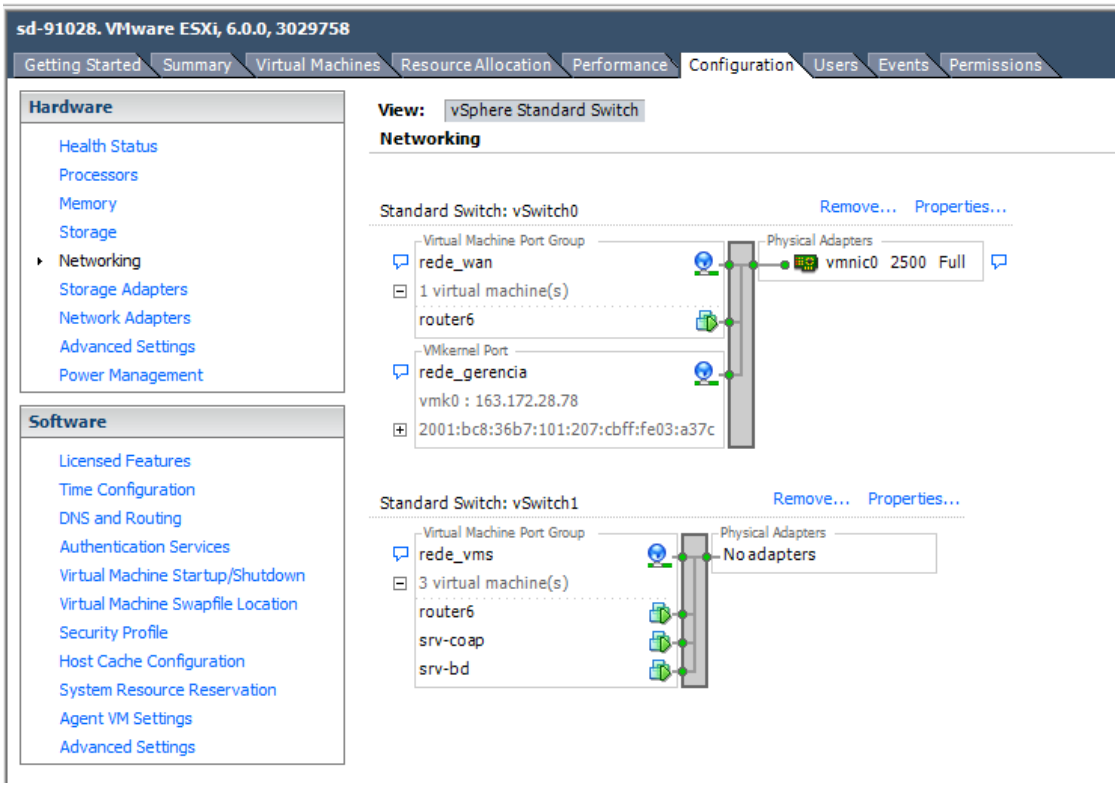


Figura 3.5: Captura de tela mostrando a configuração de rede do VMWare ESXi [autores]

O *switch* à direita do diagrama da figura é o que interliga as VMs. Por esse *switch* passam tráfegos IPv4 e IPv6 - por isso a rede é chamada de *dual stack* pois duas pilhas de protocolo são executadas simultaneamente nos mesmos *hosts* da rede, permitindo comunicação utilizando os dois protocolos. Essa configuração foi especialmente útil nos momentos de instalação iniciais das máquinas, em que pacotes foram utilizados para fazer as instalações e a instalação dos mesmos foi feita utilizando acesso à Internet em IPv4, uma vez que o IPv6 não estava configurado.

A partir das próximas seções será feito um detalhamento melhor das configurações realizadas na central.

3.5.1 Configuração dos switches virtuais

A plataforma de virtualização VMWare ESXi permite a criação de *switches* virtuais. Esses *vSwitches*, como são chamados, funcionam como seus equivalentes reais, segmentando as redes e assim isolando as interfaces de rede dos *hosts*. Nesse caso específico, os dois *vSwitches* são utilizados para separar os segmentos correspondentes à rede externa (`rede_wan`) e à rede interna (`rede_vms`), de forma que todo o tráfego de saída das máquinas virtuais seja feito através da VM que funciona como o *gateway* da rede.

A configuração dos *vSwitches* está coerente com a topologia apresentada na Figura 3.4, conforme mostrado na tela de configuração do ESXi na Figura 3.5.

Vale notar que nessa Figura há a presença de mais uma rede, chamada de `rede_gerencia`. Essa é a rede de gerência do próprio *host* virtualizador e é ligada ao primeiro *vSwitch* pois o mesmo está ligado à interface física do *host*. Isso é necessário pois, nesse caso específico, não há uma interface de rede (física) dedicada para o gerenciamento do *host* e não há possibilidade de acessar a infraestrutura física e fazer modificações uma vez que a mesma se encontra em outro país.

As demais redes são as mesmas apresentadas na Figura 3.4. Essas redes são representadas por um conjunto de portas no *vSwitch* (*Virtual Machine Port Group*), onde pode-se observar listadas as máquinas virtuais conectadas de acordo com a Figura 3.4. Ainda nessa figura é possível observar que a máquina virtual que funciona como *gateway* é a única conectada às duas redes.

3.5.2 Configuração IPv4

3.5.2.1 Roteador Dual Stack

No ambiente em que o servidor está localizado, o endereço IPv4 utilizado pelas máquinas virtuais está em uma subrede completamente diferente da subrede de gerência do *host*. Como exemplo, o endereço IP de gerência do servidor em questão pertence à rede `163.172.28.0/24`, assim como o endereço do *gateway* informado pela empresa que forneceu o serviço. Já o endereço IP válido atribuído à máquina virtual com acesso externo (a VM `router6`, no caso) é `212.83.171.5`. Isso ocorre pois o endereço IP é contratado também como um endereço de `failover` com capacidade de alocação dinâmica (a qualquer momento e a critério do contratante) a qualquer um dos *hosts* localizado em qualquer um dos *datacenters* operados pela empresa.

Por isso, a configuração de rede nesse ambiente utiliza de um artifício, que é configurar a interface de rede como se ela estivesse ligada a um enlace ponto a ponto - mesmo que ela não esteja. Para isso, é adotada a máscara de sub-rede `255.255.255.255` e um endereço de *broadcast* igual ao próprio endereço IP.

A configuração das interfaces de rede no Ubuntu pode ser feita no arquivo `interfaces`, localizado em `/etc/network` ou utilizando os utilitários `ifconfig` ou `ip address`. O trecho a seguir mostra um pedaço da configuração da interface de rede conforme as observações citadas anteriormente:

```
# Trecho do arquivo /etc/network/interfaces na VM router6
iface eth0 inet static
        address 212.83.171.5
        netmask 255.255.255.255
        broadcast 212.83.171.5
```

Além disso, para que essa configuração funcione, é preciso informar ao sistema operacional qual o endereço utilizado como *gateway padrão*, pois isso é necessário para que o sistema saiba para onde enviar o tráfego com destino à Internet. Isso foi feito através da adição manual de rotas estáticas, uma vez que o endereço de *gateway* não pode ser configurado normalmente uma vez que pertence a outro segmento de rede.

Primeiramente, o sistema foi configurado para enviar qualquer tráfego com destino ao endereço do *gateway* (`163.172.28.1`) pela interface `eth0`:

```
# route add 163.172.28.1 dev eth0
```

Então, esse endereço foi adicionado como *gateway* padrão do sistema:

```
# route add default gw 163.172.28.1 dev eth0
```

Como essas configurações duram apenas até a inicialização seguinte do sistema operacional, foi preciso torná-la modificações permanentes. Uma forma de fazer isso é adicionar ao final da configuração das interfaces (no arquivo `/etc/network/interfaces`) utilizando o parâmetro `post-up`, que executa um comando especificado após a interface de rede ser habilitada. Para evitar rotas estáticas incorretas, foi adicionada também uma configuração que as remove quando as interfaces são desativadas. Dessa forma, a configuração final da interface `eth0` passou a ser:

```
# Trecho do arquivo /etc/network/interfaces na VM router6

iface eth0 inet static
        address 212.83.171.5
        netmask 255.255.255.255
        broadcast 212.83.171.5
        post-up route add 163.172.28.1 dev eth0
        post-up route add default gw 163.172.28.1 dev eth0
        post-down route del 163.172.28.1 dev eth0
        post-down route del default gw 163.172.28.1
```

Vale observar que a configuração das rotas foi feita com o comando `route` mas também poderia ter sido feita com `ip route`.

A configuração da outra interface dessa máquina (`eth1`) é feita normalmente com a definição de um endereço estático e uma máscara de sub-rede comum (`255.255.255.0` ou `/24` na notação CIDR), permitindo até 254 máquinas virtuais nessa mesma rede.

O trecho a seguir mostra um pedaço da configuração dessa interface de rede, onde é mostrado que o endereço dessa interface (a ser utilizado como *default gateway* pelas demais máquinas) é `172.16.1.1`.

```
# Trecho do arquivo /etc/network/interfaces na VM router6
iface eth1 inet static
        address 172.16.1.1
        netmask 255.255.255.0
```

Por fim, foram criadas as regras de *firewall* para essa máquina virtual. Regras específicas de encaminhamento (`FORWARDING`) precisaram ser criadas pois trata-se do roteador dual-stack da rede e também para que ela fosse capaz de realizar o NAT necessário para a subrede IPv4. Essa configuração permite que os *hosts* pertencentes ao segmento `172.16.1.0/24` pudessem acessar a Internet por meio do IP público associado à interface `eth0` da máquina virtual (`212.83.171.5`).

De maneira análoga ao que foi feito com os objetos na seção 3.4.1.4, foi criado um arquivo contendo regras para configuração para o IPTables. O conteúdo desse arquivo, chamado `iptables.rules`, contem tanto as regras de encaminhamento quanto as regras de entrada e saída. O mesmo é apresentado a seguir:

```
*nat
:PREROUTING ACCEPT [201:11553]
:INPUT ACCEPT [3:547]
:OUTPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -o eth0 -j MASQUERADE
COMMIT

# Generated by iptables-save v1.4.21 on Sat Apr 23 15:15:07 2016
*filter
:INPUT DROP [12:3388]
:FORWARD DROP [40:3472]
:OUTPUT ACCEPT [2:80]
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -p tcp --dport 22 -j ACCEPT
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmp -j ACCEPT
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
# Completed on Sat Apr 23 15:15:07 2016
```

O primeiro conjunto de regras se refere ao NAT. O padrão em todas as cadeias do IPTables é o aceite dos dados (`ACCEPT`), pois filtros, quando necessários, podem ser aplicados na seção `*filter`. Além disso, conexões de entrada só chegarão aos servidores na subrede `172.16.1.0/24` através de redirecionamento de portas, que não foi utilizado no trabalho. O importante na seção `*nat` é a linha que efetiva o SNAT (*Source NAT*) para o tráfego com saída para a interface `eth0`, que é a linha:

```
-A POSTROUTING -o eth0 -j MASQUERADE
```

Essa linha permite que os servidores Web sejam capazes de acessar a Internet (para instalar atualizações e novos pacotes, por exemplo), o que é essencial para configuração de algumas ferramentas utilizadas na configuração da rede IPv6, que será mostrada nas próximas seções. Essa configuração faz com que na *chain* de `POSTROUTING` (regra que é processada após a decisão de roteamento ter sido tomada) ocorra a troca (`MASQUERADE`) do endereço de origem para todo o tráfego que sai da interface `eth0`.

Na seção `*filter`, são realizadas as filtragens necessárias. São autorizados somente tráfegos de entrada para gerenciamento do servidor (porta 22 para SSH) além da porta 80 e 443 para HTTP e HTTPS, respectivamente. Como grande parte do tráfego é feita em IPv6, poucas regras IPv4 existem.

Vale observar, também, que foi criada uma regra de entrada nas portas 80 e 443 (HTTP e HTTPS) ao invés de uma regra de encaminhamento. Isso foi feito pois foi tomada a decisão de usar um servidor Web como proxy reverso nessa máquina virtual afim de eliminar a necessidade de redirecionamento de portas nessa máquina virtual que funciona como *gateway*. Mais detalhes sobre isso serão apresentados na seção 3.7.1.

Além das regras do IPTables, foi preciso configurar o kernel para realizar encaminhamento de datagramas IPv4. Para isso, foi modificado o arquivo `/etc/sysctl.conf` com a linha especificada abaixo (ela precisa ser descomentada caso já esteja comentada):

```
net.ipv4.ip_forward=1
```

As modificações foram então aplicadas com:

```
# sysctl -p
```

3.5.2.2 Servidor Web / Cliente CoAP e Servidor de Banco de Dados

A configuração de rede IPv4 nos demais servidores se resume basicamente a configurar endereços estáticos nas interfaces de rede. Como a faixa escolhida para a rede interna foi `172.16.1.0` com prefixo CIDR `/24`, os endereços foram arbitrariamente escolhidos dentro dessa região. Como sempre, a configuração das interfaces foi feita no arquivo `interfaces`, podendo também ser realizada utilizando os utilitários `ip address` ou `ifconfig`. O trecho de configuração para o servidor é apresentado abaixo:

```
# Trecho do arquivo /etc/network/interfaces
iface eth0 inet static
        address 172.16.1.100
        netmask 255.255.255.0
        gateway 172.16.1.1
        dns-nameservers 8.8.8.8 8.8.4.4
```

Como pode ser observado no trecho, a novidade está na configuração dos endereços DNS. Uma vez que não há servidor DHCP distribuindo endereços de IP e nem servidores DNS, foram escolhidos os endereços dos servidores do Google (`8.8.8.8` e `8.8.4.4`).

Uma configuração análoga se aplica ao servidor de Banco de Dados, mudando apenas o endereço IP:

```
# Trecho do arquivo /etc/network/interfaces
iface eth0 inet static
        address 172.16.1.101
        netmask 255.255.255.0
        gateway 172.16.1.1
        dns-nameservers 8.8.8.8
```

Com essa configuração, caso os *hosts* precisem comunicação com a Internet usando IPv4, eles poderiam utilizar a funcionalidade de NAT da máquina virtual `router6`.

3.5.3 Configuração IPv6

3.5.3.1 Roteador Dual Stack

A configuração de IPv6 nos servidores depende primariamente da decisão de endereçamento a ser feita. O prefixo entregue pela empresa que aluga o servidor no *datacenter* na França é `2001:bc8:36b7::/48`, o que permite 65536 redes com prefixo `/64`. Assim, desse bloco de endereços, foram separados dois blocos:

- Bloco `2001:bc8:36b7:101::/64` - Rede WAN;
- Bloco `2001:bc8:36b7:102::/64` - Rede interna (`rede_vms`).

Dessa forma, a interface externa da máquina virtual `router6` recebe, por decisão de projeto, o primeiro endereço de sua faixa: `2001:bc8:36b7:101::1` e qualquer *host* na Internet pode acessar o servidor utilizando esse endereço. Analogamente, a interface interna da máquina recebe o primeiro endereço da faixa correspondente: `2001:bc8:36b7:102::1/64`. Dessa forma, as demais máquinas virtuais da rede interna receberão endereços IPv6 de escopo global nessa mesma faixa.

Tomada a decisão de endereçamento, é preciso habilitar, no kernel, o encaminhamento de datagramas IPv6, de forma parecida com o que foi feito na seção 3.5.2.1. Para isso, foi feita a modificação no arquivo `/etc/sysctl.conf` com a linha especificada abaixo (é preciso descomentar a linha caso ela esteja comentada):

```
net.ipv6.conf.all.forwarding=1
```

Então as configurações foram aplicadas com:

```
# sysctl -p
```

A configuração das interfaces de rede é feita no arquivo `interfaces`, conforme padrão das distribuições derivadas do Debian. Nesse arquivo, foi feito um incremento à configuração IPv4 anterior que consistiu em configurar de maneira estática o endereço das interfaces de rede e o prefixo (`/64`) da rede. Além disso, a configuração dos servidores DNS a serem utilizados também foi definido, conforme mostra o trecho do arquivo de configuração abaixo:

```
# Trecho do arquivo /etc/network/interfaces
auto eth0
iface eth0 inet6 static
        address 2001:0bc8:36b7:101::1
        netmask 64
        dns-nameservers 2001:4860:4860::8888 2001:4860:4860:8844
        accept_ra 2
```

Vale observar que esse trecho contem a configuração da interface externa (pertencente à `rede_wan`), com uma opção adicional, identificada por `accept_ra 2`. Essa configuração é necessária pois, uma vez que o encaminhamento está habilitado no kernel, o sistema entende que ele é o *gateway* da rede - e de fato é - e por isso ele passa a não aceitar *router advertisements* de outros roteadores da rede. Contudo, na interface externa ele deve ter o tráfego direcionado para o *gateway* da empresa de hospedagem para que o tráfego possa ser roteado para a Internet. E é para isso que serve a opção `accept_ra 2` - ela faz com que o sistema utilize os anúncios de roteadores (RAs) que chegarem a essa interface de rede. Dessa forma, a máquina passa a ter uma rota de saída para a Internet.

Para a interface de rede interna (`rede_vms`), o procedimento é semelhante, pois bastou definir um endereço IPv6 estático na configuração. Apesar de não ser necessário nesse caso, os endereços dos servidores DNS também foram definidos para essa interface, conforme mostrado no trecho abaixo:

```
# Trecho do arquivo /etc/network/interfaces
auto eth1
iface eth1 inet6 static
        address 2001:0bc8:36b7:102::1
        netmask 64
        dns-nameservers 2001:4860:4860::8888 2001:4860:4860:8844
```

Com os endereços corretamente configurados, é preciso configurar como os endereços serão distribuídos para a rede interna (`rede_vms`). Três formas eram escolhas possíveis:

1. Configuração manual dos endereços nos servidores;
2. Configuração automática dos endereços utilizando DHCPv6;
3. Configuração automática dos endereços baseada em SLAAC (configuração sem estado automática de endereços, usando para isso RADV).

A primeira opção é válida quando se possui poucos servidores, porém, assim que a quantidade de servidores aumenta, essa opção se torna menos viável e mais sucetível a erros. A segunda opção é uma opção válida, mas exige o uso de um servidor DHCP na rede para coordenar a alocação dos endereços - apesar de ser possível, está fora do escopo do trabalho. Por fim, a última opção foi escolhida - ela permite que os *hosts* se configurem automaticamente sem o uso de um servidor DHCP centralizado, alternativa igual à utilizada para os objetos da rede (seção 3.4.1.3).

Para isso, foi necessário instalar e configurar daemon `radvd`, que faz anúncios de roteador (`router advertisements`) para a rede. Essa aplicação foi instalada através do gerenciador de pacotes do Ubuntu com:

```
# apt-get install radvd
```

A configuração desse *daemon* é feita em `/etc/radvd.conf`. O arquivo utilizado é apresentado abaixo:

```
# Arquivo /etc/radvd.conf

interface eth1 {
    AdvSendAdvert on ;
    MaxRtrAdvInterval 30;

    prefix 2001:bc8:36b7:102::/64 {
        AdvOnLink on;
        AdvAutonomous on;
    };
};
```

A explicação dos parâmetros utilizados foi apresentada na seção 3.4.1.3, pois esse *daemon* também está em execução no *gateway* dos objetos. Em resumo, ele anuncia o bloco de IPs configurado em `prefix` (`2001:bc8:36b7:102::/64`) para a rede `rede_vms`. Ao receber os anúncios feitos por esse servidor, os demais servidores da rede irão utilizar o identificador EUI-64 (montado com base no MAC Address do *host*) para completar o prefixo anunciado.

As regras de *firewall* criadas precisam apenas autorizar o encaminhamento de tráfego UDP com porta `5683` (a porta padrão do protocolo CoAP), tráfego ICMPv6 (necessário para o uso de mensagens de *router advertisements*, cuja configuração será mostrada nas próximas seções), tráfego UDP nas portas 546 e 547 (necessário para funcionamento do DHCPv6, também mostrado nas próximas seções) e tráfego TCP na porta de destino 22 (padrão do protocolo SSH), para que seja possível fazer acesso remoto e gerenciar o servidor à distância.

Além disso, como os servidores pertencentes à rede `rede_vms` acessarão a Internet utilizando IPv6 através do *gateway*, é preciso autorizar o encaminhamento de pacotes para tráfego HTTP (portas TCP 80 e 443) e DNS (porta 53 tanto TCP quanto UDP). Como essa máquina também precisa de receber tráfego HTTP de entrada, as regras que dizem respeito à HTTP são criadas tanto na cadeia de encaminhamento (`FORWARD`) quanto na cadeia de entrada (`INPUT`). Assim, o arquivo de regras `ip6tables.rules` foi criado em `/etc/network` com o seguinte conteúdo:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmpv6 -j ACCEPT
-A INPUT -p tcp --dport 22 -j ACCEPT
-A INPUT -p udp --dport 547 -d ff00::/8 -j ACCEPT
-A INPUT -p udp --dport 546 -d ff00::/8 -j ACCEPT
-A INPUT -p udp --dport 5683 -j ACCEPT
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmpv6 -j ACCEPT
-A FORWARD -p udp --sport 5683 -j ACCEPT
-A FORWARD -p udp --dport 5683 -j ACCEPT
-A FORWARD -p tcp --dport 80 -j ACCEPT
-A FORWARD -p tcp --dport 443 -j ACCEPT
-A FORWARD -p tcp --dport 53 -j ACCEPT
-A FORWARD -p udp --dport 53 -j ACCEPT
-A FORWARD -p tcp --dport 22 -j ACCEPT
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
```

É preciso aplicar as regras de *firewall* automaticamente. Novamente, as regras foram configuradas para serem aplicadas no momento em que uma interface de rede é habilitada. Assim, o arquivo `interfaces` localizado em `/etc/network` foi modificado para incluir a linha abaixo:

```
pre-up ip6tables-restore < /etc/network/ip6tables.rules
```

Assim, o trecho do arquivo de configuração das interfaces ficou como mostrado abaixo:

```
auto eth1
iface eth1 inet6 static
        address 2001:0bc8:36b7:102::1
        netmask 64
        dns-nameservers 2001:4860:4860::8888 2001:4860:4860:8844
        pre-up ip6tables-restore < /etc/network/ip6tables.rules
```

3.5.3.2 Servidor Web e Cliente CoAP

Uma vez que os anúncios de rota são feitos pelo *gateway* (máquina virtual `router6`), os demais servidores podem ser configurados para obter configuração de endereço automaticamente. Uma vez que o prefixo não muda e o identificador EUI-64 não muda quando as extensões de privacidade [IETF 2007] não estão ativadas (devido ao MAC Address não mudar), o endereço será fixo mesmo que não haja uma entidade central como um servidor DHCP. A única configuração manual que precisa ser feita é a configuração dos servidores DNS, muito embora essa configuração também possa ser automatizada com o uso de um servidor DHCP para distribuir exclusivamente essa informação.

Dessa forma, a configuração IPv6 da interface de rede do servidor ficou como apresentado a seguir:

```
auto eth0
iface eth0 inet6 auto
    dns-nameservers  2001:4860:4860::8888 2001:4860:4860:8844
```

Basta desabilitar e habilitar novamente a interface de rede:

```
# ifdown eth0
# ifup eth0
```

Em seguida, verificar o endereço IPv6 com o comando `ifconfig`, a saída incluía o endereço IPv6, conforme mostrado abaixo:

```
eth0      Link encap:Ethernet  HWaddr 00:0c:29:7c:ae:38
          inet addr:172.16.1.100  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: 2001:bc8:36b7:102:20c:29ff:fe7c:ae38/64 Scope:Global
          inet6 addr: fe80::20c:29ff:fe7c:ae38/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3529144 errors:0 dropped:49 overruns:0 frame:0
          TX packets:3135420 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1373528318 (1.3 GB)  TX bytes:1220411883 (1.2 GB)
```

No caso desse servidor, o endereço IPv6 configurado por RADVD é:

```
2001:bc8:36b7:102:20c:29ff:fe7c:ae38
```

Para o servidor de banco de dados, a configuração é idêntica à que foi feita para o servidor Web. O endereço será configurado automaticamente devido ao MAC Address diferente do servidor. Assim, o endereço configurado no servidor de banco de dados foi:

```
2001:bc8:36b7:102:20c:29ff:fe4b:6a41
```

Vale observar, contudo, que como o servidor de banco de dados não precisa estar visível na Internet, ele não precisaria de um endereço IP válido. Isso acontece pois o servidor de aplicação (o servidor Web no caso) é o único que deve acessar o banco de dados diretamente, assim, bastaria a configuração automática para IPv6 com endereços do tipo link-local, ou até mesmo apenas a configuração IPv4. A configuração do servidor de banco de dados com endereço de escopo Global foi feita para fins de demonstração.

3.6 PROGRAMAÇÃO DOS OBJETOS

3.6.1 Arquivo de configuração

Para evitar inserir diretamente no código os parâmetros, foi criado um arquivo de configuração para o código em Python responsável pela leitura dos sensores. Esse arquivo de configuração está em formato YAML e é apresentado abaixo:

```
sensors:
    sleep: 10
    pins:
        temperature: 11

files:
    sensors_out:
        temperature: "temperature.json"
        humidity: "humidity.json"

interface: "lowpan0"
```

Nesse arquivo, a seção `sensors` especifica parâmetros relevantes para a leitura dos sensores: o parâmetro `sleep` especifica o intervalo (em segundos) entre leituras sucessivas dos sensores enquanto a subseção `pins` contem os pinos que devem ser lidos para cada um dos *resources* especificados. Nesse exemplo, apenas `temperature` é especificado pois o pino é compartilhado com o pino de leitura de umidade.

A seção `files` contem parâmetros relevantes para a configuração dos arquivos que são salvos contendo as informações lidas pelos sensores. Cada resource deve conter um nome de arquivo - aqui a escolha foi utilizar o nome do *resource* no nome do arquivo.

Por fim, o parâmetro `interface` especifica a interface utilizada para comunicação. Ela é usada pelo módulo `netifaces` para obter o endereço IPv6 global do objeto - utilizado como identificador do objeto no banco de dados e incluído em cada uma das mensagens enviadas à central.

3.6.2 Identificação dos objetos

Durante o desenvolvimento, haviam duas possibilidades para a identificação dos objetos na central:

1. Obter o endereço de IP de origem de cada mensagem - na central;
2. Fazer o objeto identificar seu endereço de IPv6 válido e incluir nas mensagens enviadas à central.

O segundo método foi escolhido pela facilidade de manipular essas informações utilizando uma biblioteca já existente para Python (chamada `netifaces`). Essa biblioteca possui uma função que, dada uma *string* contendo o nome de uma interface, retorna um conjunto de endereços.

Então, uma função foi criada para encontrar um endereço IPv6 global associado a uma interface especificada. Primeiramente, a interface deve ser especificada no arquivo de configuração (seção 3.6.1), assim a função apresentada abaixo pode ser chamada utilizando como parâmetro o nome da interface lido do arquivo de configuração:

```
def get_if_global_6address(if_name):
    try:
        # Obtem os enderecos IPv6 da interface especificada
        # um conjunto de enderecos sera retornado - ao menos um global e um
        # link-local
        for if_info in netifaces.ifaddresses(if_name)[netifaces.AF_INET6]:
            # buscamos um endereco global (nao iniciado em fe80)
            if if_info["addr"].split(":")[0] != "fe80":
                return if_info["addr"]
    except ValueError:
        print "erro, interface de rede especificada eh invalida"
```

A função acima é chamada sempre que o código precisa gerar os objetos JSON contendo os dados dos sensores. Dessa forma, o identificador do objeto (`object-id`, conforme será mostrado mais adiante) é inserido em todos os objetos que são enviados à central. Para garantir que um endereço IPv6 global seja enviado ao invés de um endereço `link-local`, a *string* contendo o endereço IPv6 é analisada: é selecionado o primeiro endereço em que os primeiros quatro caracteres não correspondam a `fe80`. Daqui surge uma restrição: a interface utilizada deverá possuir apenas um endereço global.

3.6.3 Sensores

3.6.3.1 Sensor de Temperatura e Umidade

Conforme apresentado na seção 2.5.1.1, o DHT11 é um sensor de temperatura e umidade que se comunica utilizando um único pino de dados. Após receber um pedido por esse pino, o controlador atrelado ao sensor retorna 40 bits de dados, dos quais:

- 8 bits são a parte inteira da leitura de umidade;
- 8 bits são a parte decimal da leitura de umidade;
- 8 bits são a parte inteira da leitura de temperatura;
- 8 bits são a parte decimal da leitura de temperatura;
- 8 bits de *checksum* para verificação de erros.

Foi utilizada uma biblioteca (do grupo *Adafruit*) para realizar a abstração sobre o formato dos dados trocados. Isso permitiu maior facilidade na aquisição dos dados, pois a biblioteca faz todo o trabalho de lidar com os sinais trocados entre o sensor e o Raspberry Pi. Essa facilidade é mostrada no trecho resumido do código de leitura das medições do sensor, mostrado na seção de apêndices I.1.1.

3.6.3.2 Sensor de Luminosidade

O trecho de código em Python responsável pela leitura do fotoresistor através de um conversor analógico digital é apresentado na seção I.1.2.

3.6.3.3 Integração com o Actinium

Os dados dos sensores são obtidos utilizando Python e um conjunto de bibliotecas *opensource* desenvolvidas pela comunidadde, conforme mostrado nas seções 3.6.3.1 e 3.6.3.2. Porém, o servidor de aplicação utilizado foi desenvolvido em Java e possui um interpretador pois os códigos que definem as ações dos objetos são na verdade escritos em Javascript.

A forma encontrada para realizar essa integração é utilizar o sistema de arquivos do objeto. Dessa forma, os códigos dos sensores em Python foram programados para gravar periodicamente os dados em arquivos em um local pré-determinado no disco. Esses arquivos são então periodicamente lidos pelo servidor de aplicação (Actinium), que de tempos em tempos notifica a central através de mensagens CoAP. Essas mensagens são respostas à solicitação de *observe* original feita pela central.

O formato de dados escolhido foi o JSON, pela facilidade de integração com os códigos em Javascript. Inicialmente, um dicionário é criado em Python e, atravéz de mais uma biblioteca, esse dicionário é convertido em um objeto Javascript (representado em JSON). Esse objeto é gravado em um arquivo `.json` com o nome do *resource* que representa. Por exemplo: os dados de temperatura são gravados em um arquivo chamado `temperature.json` pois o nome do *resource* instalado no Actinium é `temperature`. A mesma lógica se aplica para a umidade, em que o nome do arquivo é `humidity.json`.

O trecho de código em Python responsável por gerar esses objetos é apresentado na seção I.1.3. Os arquivos `temperature.json`, `humidity.json` e `lights_read.json` são apresentados abaixo:

```
{"object-id": "2001:1291:200:85b2:1ac0:ffee:1ac0:ffee", "sensor-values": {"pins": 11, "values":
    27.0}, "resource": "temperature", "time": 1475952631.663539}

{"object-id": "2001:1291:200:85b2:1ac0:ffee:1ac0:ffee", "sensor-values": {"pins": 11, "values":
    16.0}, "resource": "humidity", "time": 1475952578.96591}

{"object-id": "2001:1291:200:5de::2", "sensor-values": {"pins": 3, "values": 6.62}, "resource":
    "lights_read", "time": 1477685621.967546}
```

Esse arquivos carregam os seguintes dados:

- Identificador do objeto (`object-id`): um endereço IPv6 global obtido utilizando a biblioteca `netifaces` conforme mostra a seção 3.6.2;
- Os valores lidos pelos sensores (`sensor-values.values`) e o respectivo pino de onde o valor foi lido no Raspberry Pi (`sensor-values.pins`). Os nomes estão no plural para caso múltiplos valores existam para um mesmo *resource*, situação na qual os campos passam a ser *arrays* de valores;
- O nome do *resource*;
- O instante de tempo (*timestamp*) da leitura do valor, (campo `time`).

O Actinium foi carregado com um código para cada um dos *resources* lidos. O passo-a-passo do *upload* do código no servidor Actinium se encontra no Apêndice. Os dois códigos são muito parecidos, a diferença entre eles está o caminho para o arquivo lido, uma vez que diferentes arquivos são gerados para diferentes *resources*.

É muito importante ressaltar que havia a necessidade do método `OBSERVE` do CoAP ser utilizado, pois isso elimina a necessidade de *polling* aos objetos pois esse modelo permite que a central "assine"os *resources* dos objetos apenas na primeira solicitação, fazendo com que os objetos possam enviar suas respostas quando for conveniente.

No caso, para simplificar o funcionamento e facilitar os testes, um temporizador foi colocado nos objetos para que a cada 5 minutos os dados dos sensores fossem enviados. No entanto, outras estratégias poderiam ser adotadas, como o envio da informação somente quando ela for alterada ou somente quando ela superar limiares estabelecidos pelo usuário.

O código em Actinium responsável por tratar requisições do tipo `GET` que chegam aos objetos e responder periodicamente às solicitações de `OBSERVE` é apresentado na seção I.2.

3.7 CONFIGURAÇÃO DOS SERVIDORES NA CENTRAL

A central foi pensada para se parecer com um ambiente real. Um *host* físico virtualiza três máquinas virtuais, que são os servidores. Desses servidores, um é o roteador pilha dupla da rede, que é utilizado como *gateway* pelos demais servidores.

As máquinas virtuais foram divididas de acordo com sua função básica na rede:

- Servidor Web e Cliente CoAP (`srv-coap`);
- Servidor de banco de dados (`srv-bd`);
- Gateway e Web Proxy Reverso (`router6`).

Dessa forma, numa situação em que é necessário aumentar a capacidade, é possível mover as máquinas virtuais para outros *hosts*, aumentar os recursos disponíveis para cada máquina (CPU, RAM e disco) e até mesmo replicar os servidores utilizando múltiplas VMs com a mesma funcionalidade, o que além de aumentar a capacidade também pode aumentar a disponibilidade dos serviços.

Cada uma das máquinas listadas será descrita a seguir.

3.7.1 Servidor Web

O servidor Web é quem exibe a página de controle da central para os usuários (detalhes de funcionamento serão descritos na seção 3.7.3). Por consequência, ele também é o cliente CoAP na comunicação com os objetos, que são servidores de aplicação executando Actinium.

O servidor Web é um simples servidor executando uma aplicação NodeJS que, além de servidor de conteúdo estático, realiza operações no banco de dados e funciona como cliente CoAP através de uma biblioteca da comunidade. Esse servidor é quem de fato inicia as relações de `OBSERVE` com os objetos e permite a consulta aos dados pelos usuários através de uma interface Web.

Uma vez que a aplicação Web não depende de um servidor Web externo, a aplicação em si executa seu próprio servidor. Basta colocar a aplicação em execução e configurar o servidor de proxy reverso (próximas seções) para se comunicar com a aplicação na porta correta.

3.7.2 Proxy Reverso HTTP

Durante a configuração da rede foi mostrado que os servidores são capazes de funcionar com pilha dupla. Porém, os servidores Web e de banco de dados possuem endereços IPv4 locais (inválidos para roteamento na Internet), utilizando NAT para acesso à Internet.

Em IPv6, não há NAT, portanto o servidor Web está diretamente acessível através da Internet, situação em que o servidor `router6` atua apenas como roteador IPv6. Porém, em IPv4, a presença de NAT exigiria redirecionamento de portas além da simples função de roteamento. Para evitar que isso ocorresse (e também ganhar funcionalidades), foi adotada uma outra solução, que consiste em aproveitar a máquina `router6` para acrescentar a função de proxy reverso.

Com o uso de proxy reverso, as requisições HTTP que seriam encaminhadas ao servidor Web passam a ser encaminhadas ao servidor proxy. Esse servidor, então, realiza as requisições HTTP com o chamado servidor *upstream*, que é o servidor Web. Como a comunicação IPv4 entre o servidor proxy e o servidor Web já existe usando endereços locais, não há necessidade de utilizar redirecionamento de portas.

Essa abordagem possui algumas outras vantagens, entre as quais destacam-se:

- Cache de conteúdo estático pelo Proxy;
- Amortecimento de ataques de negação de serviço (SYNFLOOD), uma vez que o proxy passa a receber todos os estabelecimentos de conexão passando apenas as requisições válidas aos servidores de *upstream*;
- Possibilidade de implementação de balanceamento de carga em que diferentes requisições são repassadas a diferentes servidores Web de forma a distribuir a carga de processamento entre eles, aumentando assim a capacidade;
- Estabelecimento de conexões HTTP sobre TLS (HTTPS) quando a criptografia não pode ser implementada diretamente no servidor Web. Isso aumenta a segurança em relação à utilização de HTTP simples, que não implementa criptografia.

Na arquitetura proposta, o servidor proxy possui a função de cache de conteúdo estático bem como a função de terminação de conexões HTTPS. Assim, o redirecionamento de portas na máquina `router6` foi evitada, já que as requisições são encaminhadas ao servidor Web pelo proxy reverso.

O servidor de Proxy Reverso escolhido foi o NGINX, que também poderia ser utilizado como servidor Web. O NGINX foi configurado de forma a encaminhar requisições HTTP para o servidor Web (de aplicação), ou seja, o servidor de *upstream* além de armazenar em logs todas as solicitações, fazer *cache* de conteúdo estático e funcionar como terminação de SSL (para conexão HTTPS com os usuários). A configuração do NGINX é dividida em duas partes é definida na seção I.3 dos apêndices.

3.7.3 Servidor de Banco de Dados

O servidor de banco de dados está no mesmo segmento de rede do servidor Web. Apesar de possuir um endereço IPv6 válido para acesso à Internet, a ideia é que esse serviço não seja acessado diretamente pela Internet, mas sim apenas pelo servidor Web.

Essa decisão foi tomada para evitar que vulnerabilidades no gerenciador do banco de dados sejam expostas à Internet. Com os acessos concentrados no servidor Web, a exploração de vulnerabilidades ainda é possível, porém é dificultada, pois as vulnerabilidades tem de existir em conjunto com falhas de segurança na aplicação.

Uma configuração importante foi feita na instalação padrão do MongoDB: permitir que ele passe a escutar por conexões provenientes de outros *hosts* além de *localhost*. Em um ambiente de produção, é desejável também a criação de um usuário a ser utilizado pela aplicação, medida que limita as permissões da aplicação Web a um banco de dados específico. Dessa forma, se a aplicação for comprometida, os danos serão limitados a um único banco de dados, não afetando bancos utilizados por outras aplicações, caso existam. Como o projeto possui apenas um banco de dados e o ambiente não é compartilhado com nenhuma outra aplicação, essa segunda configuração não foi realizada.

A primeira configuração foi feita editando o arquivo de configuração do MongoDB na seção `net`. O parâmetro `bindIp` foi modificado para `0.0.0.0`, o que autoriza conexões provenientes de todos os endereços. Isso permitiria o acesso proveniente de qualquer *host* da Internet, não somente do servidor Web. Porém, deve-se lembrar da configuração feita no *firewall* da máquina *router6*, que bloqueia esse tipo de conexão na CHAIN `FORWARD`.

3.8 DESENVOLVIMENTO DA APLICAÇÃO NA CENTRAL

O objetivo do trabalho era mostrar o funcionamento de uma rede para IoT, uma prova de conceito para demonstrar o funcionamento da nova pilha de protocolos (*constrained*). Para alcançar esse objetivo, uma pequena e simples aplicação foi desenvolvida afim de gerar os dados necessários para se obter tráfego e também como forma de monitorar o funcionamento dos sistema através dos dados gerados pelos objetos.

A aplicação é desenvolvida em duas partes principais: uma parte que executa no navegador do usuário quando ele acessa o sistema, chamada de *front-end* e outra parte que executa nos servidores da central, chamada de *back-end*. A função do *front-end* é mostrar as informações provenientes do *back-end* aos usuários ao mesmo tempo em que recebe as ações dos usuários e dispara ações correspondentes no *back-end*, quando aplicável.

Na central, o código em execução trata de manter de forma simplificada um estado de todos os objetos previamente cadastrados no sistema. Junto com cada cadastro, há uma flag no banco de dados indicando *resources* dos objetos aos quais foram solicitados previamente uma relação de `OBSERVE`. Isso é necessário pois os objetos não persistem dados sobre seu estado, ou seja, uma vez que eles são desligados a listagem de *resources* em relação de `OBSERVE` com a central é perdida. Dessa forma, quando um objeto é religado por alguma falha de alimentação, a central é capaz de perceber e refazer a solicitação.

A central é capaz de perceber quando um objeto tornou-se indisponível através dos intervalos de envio de informações. Por padrão, tanto os objetos quanto a central são configurados com intervalos de 5 minutos, sendo que na central o tempo limite desde a última informação recolhida é o dobro disso. Assim, de cinco em cinco minutos é executada uma rotina na central que verifica o instante de tempo em que a última informação chegou à central e, caso a diferença entre o instante de verificação e o instante de chegada da informação seja superior a dez minutos, a central supõe que o objeto ficou indisponível e foi religado por algum motivo, e assim uma nova relação de `OBSERVE` é realizada.

Ao redor desse núcleo básico, estão as funções adicionais realizadas pelo *back-end* que são serviços oferecidos ao *front-end*. Todos esses serviços funcionam utilizando requisições HTTP do tipo `GET` ou `POST` e as respostas são enviadas em formato JSON. Abaixo é feita uma breve descrição de cada um desses serviços:

- Retornar uma listagem de objetos cadastrados na central. É importante ressaltar que os objetos precisam ser pré-cadastrados no banco de dados para que isso funcione;
- Retornar uma listagem de *resources* disponíveis, dado o identificador (endereço IPv6) do objeto como parâmetro da requisição. Essa requisição faz a central se comunicar diretamente com o objeto utilizando CoAP;
- Retornar uma listagem de *resources* em conjunto com uma informação que determina se o *resource* está sob observação da central para um determinado objeto passado como parâmetro da requisição;
- Ativar ou desativar, na central, a necessidade de estabelecimento de relações de `OBSERVE`. Essa ação altera diretamente o resultado retornado no item anterior e afeta as verificações periódicas realizadas pela central;
- Retornar a série de medições realizadas para um determinado objeto e *resource* especificados como parâmetros da requisição.

Cada uma dessas requisições é feita com base em uma ação do usuário realizada na parte do *front-end*. Na tela inicial, por exemplo, existe uma lista de objetos (que utiliza o primeiro serviço listado acima), sendo que para cada objeto é exibido uma lista de *resources*, obtida com a segunda requisição listada acima.

Cada *resource* é exibido na lista de maneira diferente se uma relação de `OBSERVE` já foi definida para o mesmo. Essa informação é obtida através da terceira requisição. Quando o usuário clica sobre um *resource* da lista que ainda não foi clicado anteriormente, a quarta requisição da lista acima modifica no banco de dados ativando a *flag* que ativa a necessidade de estabelecer uma relação de `OBSERVE`. Por fim, em uma segunda tela, quando o usuário deseja visualizar os gráficos para um determinado objeto, a última requisição da lista é utilizada acima. Nesse caso, como a requisição é válida para um objeto e *resource* específico, é preciso combinar a última requisição com as anteriores.

O usuário interage com uma interface Web simples construída utilizando Javascript (e o *framework* AngularJS) e os estilos pré-definidos por um *template* feito em Bootstrap. Algumas bibliotecas extras contendo diretivas espefícias para AngularJS foram utilizadas, como a Chart.JS, que permite a criação de gráficos em HTML5 usando elementos do tipo *Canvas*. Os dados mostrados na página são trocados com o *back-end* utilizando a notação JSON.

Por sua vez, a central executa uma aplicação em NodeJS, que ao mesmo tempo que serve os conteúdos estáticos para montagem das páginas, realiza as operações de leitura/escrita no banco de dados (MongoDB) e a comunicação com os objetos utilizando uma biblioteca que implementa o protocolo CoAP. Assim, fornece os serviços mostrados na listagem anterior.

Uma breve descrição das funções principais realizadas na central é feita nas próximas seções.

3.8.1 Estabelecimento das relações de OBSERVE do CoAP

Como mencionado anteriormente, a função principal da central é monitorar os dados enviados pelos objetos e se certificar que esses voltem a enviar dados quando são religados. Por isso, essa parte do código da central é chamado de monitor.

Como visto na seção 2.1.5, uma requisição do tipo `OBSERVE` é aquela que precisa ser feita uma única vez para que sucessivas repostas sejam enviadas de volta. Isso evita que a central tenha de realizar *polling* por todos os objetos, o que seria um grande desperdício de recursos nos servidores. Com esse modelo, tudo que os objetos tem que fazer é aguardar por requisições e registrá-las conforme chegam. Esse registro é usado para mandar as respostas para a central, pois o objeto armazena quem fez a requisição e também qual *resource* foi solicitado. As respostas são enviadas de acordo com o que foi programado nos objetos: periodicamente ou após a ocorrência de certas mudanças específicas.

No caso da aplicação desenvolvida, as respostas são enviadas periodicamente, pois essa característica é aproveitada pelo funcionamento do monitor da central, que usa as informações de periodicidade para julgar o estado dos objetos.

Assim, o monitor estabelece as relações de `OBSERVE` realizando requisições para os objetos (que são requisições do tipo `GET` com a *flag* `OBSERVE` ativadas). Os objetos, por sua vez, registram essas relações e passam a enviar as respostas a cada 5 minutos (por padrão) para a central que possui um método pronto para ser chamado cada vez que uma resposta chegar. Esse método lê a informação que chegou (já no formato JSON) e a armazena no banco de dados. Como o banco de dados MongoDB é utilizado, o objeto no formato JSON pode ser salvo diretamente no objeto, sem necessidade de muitas transformações.

3.8.2 Tratamento das respostas dos objetos

Conforme mencionado anteriormente, a central possui um método pronto que é chamado a cada resposta (CoAP) chega à central. As respostas são formadas por uma mensagem no formato JSON, que a central copia e salva no banco de dados, sem nenhum tipo de transformação. Com isso, os dados passam a estar disponíveis para exibição nos gráficos. Além disso, a central pode usar o tempo dessa resposta para determinar se o objeto foi desligado ou não (se nenhuma outra resposta chegar após dez minutos).

3.8.3 Verificação periódica de estado dos objetos

A verificação periódica do estado dos objetos é a parte mais complicada da aplicação que executa na central. Quando a aplicação é inicializada, ela define um temporizador que é chamado a cada aproximadamente cinco minutos (por padrão). A cada vez que esse tempo expira, é chamada uma grande rotina que é nomeada de monitor. A seguir é feita uma breve descrição dos passos executados pelo monitor:

1. É feita uma busca no banco de dados por todos os *resources* que possuem a *flag* `observing` definida como `true`. Essa flag é a que indica que uma relação de `OBSERVE` deve ser mantida com o objeto. Esse resultado é mantido em memória para comparação;
2. Então, para cada um dos objetos, é feita uma requisição que retorna o último tempo de medição de cada um dos *resources* de cada um dos objetos. Então, caso esse *resource* esteja na lista salva em memória do passo anterior, o tempo desse *resource* é subtraído do horário atual do sistema;
3. Caso a diferença de tempo calculada seja maior que dez minutos, uma nova requisição CoAP `OBSERVE` é realizada. Vale observar aqui que a central não é capaz de diferenciar se o objeto já foi ou não religado (ou voltou à ter conectividade): caso o objeto não esteja disponível, essa requisição será perdida e na próxima verificação do monitor todo o processo irá se repetir para esse objeto.

3.8.4 Desenvolvimento do Bot para Telegram

O Telegram oferece dois tipos de APIs para os desenvolvedores. A primeira é uma API para desenvolver o seu próprio aplicativo de mensagens. A segunda API, e a mais interessante para esse projeto, é o desenvolvimento de *bots* que poderão interagir com os usuários que possuem o Telegram como aplicativo de mensagens. Com essa API é possível criar canais em que os usuários podem se inscrever e interagir com o seu *bot*.

O objeto do *bot* seria uma ferramenta complementar para os usuários conferirem se os objetos estão ativos através do Telegram. Esse *bot* retorna os objetos ativos e o último valor medido pelos seus sensores.

O Telegram (e membros da comunidade) oferecem a API em várias linguagens de programação. Para ser compatível com a linguagem utilizada pela aplicação nesse projeto foi utilizada a biblioteca `node telegram bot api` para NodeJS.

A criação do canal que os usuários irão utilizar para ter acesso ao *bot* é realizada por um *bot* do Telegram, chamado @*botfather*, bastando seguir o *wizard* de criação. No final desse passo-a-passo é gerado um `TOKEN`. Ele é exclusivo ao canal e será utilizado na programação do *bot*.

Para estabelecer a comunicação do usuário com os servidores do Telegram e assim conseguir comunicação com o *bot* foram utilizados *webHooks*. Como essa opção só aceita uma comunicação segura, por meio de HTTPS, foi necessária a utilização de certificado SSL na comunicação. Essa configuração foi explicada na seção 3.7.2. É importante lembrar que a aplicação está atrás de um *proxy* reverso e os devidos cuidados com quais portas estão sendo utilizadas para anunciar o serviço devem ser tomados.

Já a programação do *bot* é bem direta: são programadas funções para cada comando disponibilizado para o usuário. Os comando são representados com a adição do caracter "". O primeiro comando que deve ser tratado é o comando "start"que é enviado a aplicação toda vez que um usuário novo entra no canal. Nessa primeira interação é importante salvar a identificação do usuário (*telegram ID*) que é utilizada para encontrar os objetos específicos desse usuário no banco de dados.

Outro comando importante de se tratar é o comando *help* que retorna os comandos disponíveis para o usuário. O código responsável por isso é utilizado como amostra de código para consumo da API de bots e é apresentado na seção I.4.

4 RESULTADOS

4.1 VISÃO GERAL DA REDE

Finalmente, todos os elementos da rede que foram descritos ao longo do trabalho podem ser vistos na Figura 4.1.

Nessa Figura, na parte superior esquerda é possível observar o Raspberry Pi ligado diretamente à Internet utilizando Wi-Fi. Nesse caso, o túnel 6in4 é fechado diretamente no próprio Raspberry, de forma que ele foi capaz de acessar a Internet IPv6. Esse é o Raspberry ligado com o sensor de iluminação. Na parte inferior esquerda é possível visualizar um Raspberry Pi funcionando como *gateway* e outro funcionando com os sensores de temperatura e umidade. Nessa situação o *gateway* é tanto o ponto onde o túnel 6in4 é fechado quanto a conversão entre 6LoWPAN e Wi-Fi é feita. Ambos estão ligados à uma rede Wi-Fi doméstica tradicional, formada por um "Roteador Wireless"comum, aparelho que agrega as funções de roteador, Access Point Wi-Fi e switch Ethernet.

O tráfego de ambas as redes é feito em IPv4 utilizando o protocolo AYIYA, que encapsula os datagramas IPv6 dentro de datagramas UDP que são carregados em IPv4 até chegar ao fim do túnel 6in4 (*SIXXS Tunnel Endpoint*). Então, os datagramas são desencapsulados e roteados na Internet IPv6 normalmente até chegar ao datacenter onde se encontram os servidores configurados para funcionar em IPv6.

4.2 RESULTADOS DA MONTAGEM DO HARDWARE

As fotos da montagem que utilizam 6LoWPAN são apresentadas nas Figuras 4.2 e 4.3.

A outra montagem que utiliza apenas Wi-Fi não possui o módulo de rádio, mas sim apenas um fotoresistor com um conversor analógico para digital, conforme mostra a Figura 4.4.

4.3 FUNCIONAMENTO DA REDE NOS OBJETOS

Conforme mostrado na seção 3.4, foram feitas três configurações principais: no objeto, foi feita a configuração da interface de rede 6LoWPAN. No *gateway*, foi feita a configuração do *daemon* RADV para configuração automática de endereços (SLAAC) dos objetos da rede local bem como as configurações de encaminhamento e *firewall* para permitir o roteamento entre a rede e a Internet. Além disso, no *gateway* também foram feitas as configurações do túnel IPv6.

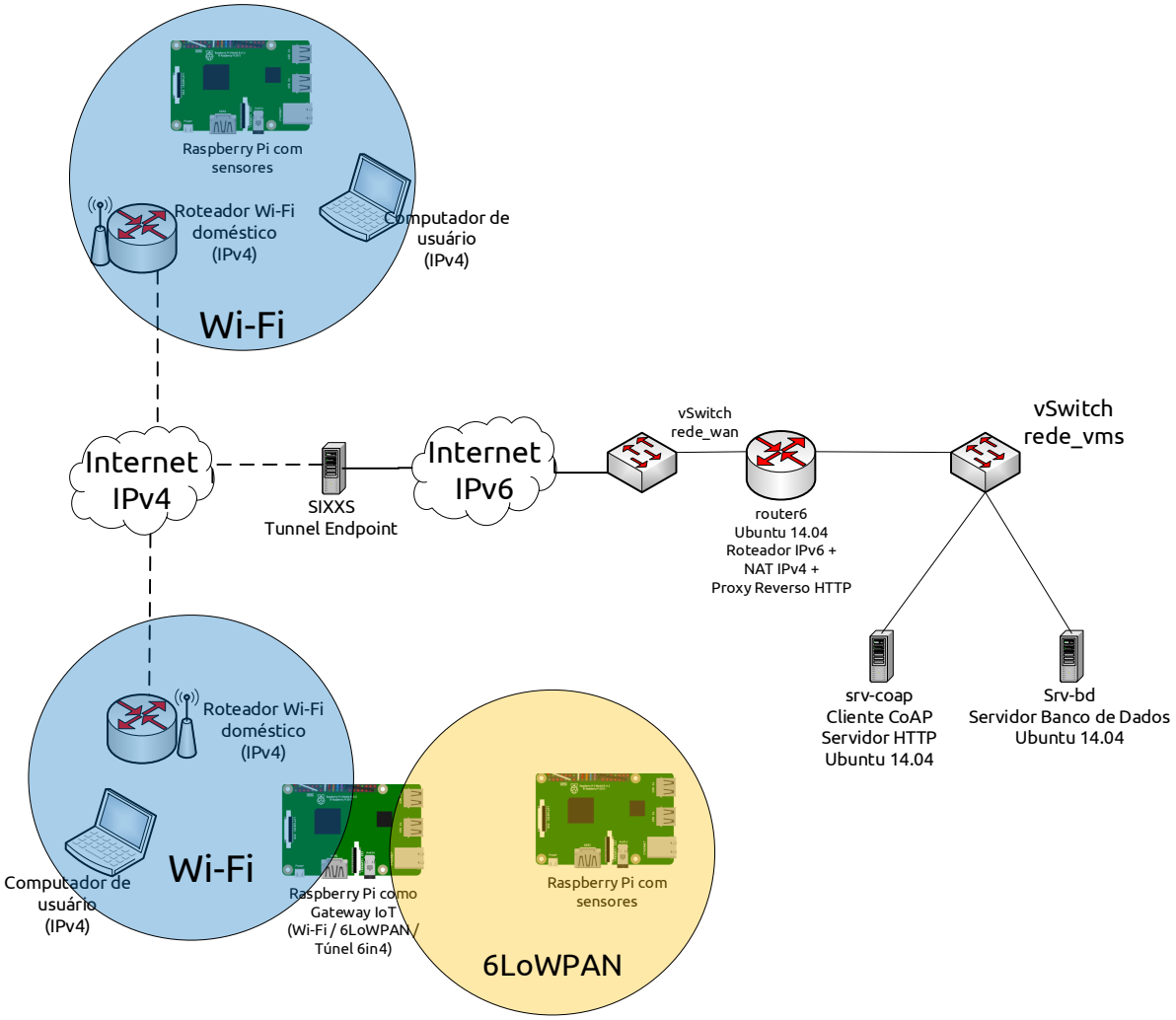


Figura 4.1: Diagrama de rede completo mostrando todos os elementos do trabalho [autores]

O sucesso da configuração automática de endereços utilizando a interface 6LoWPAN é mostrada na Figura 4.5, onde o comando `ifconfig` foi executado exibindo a configuração da interface `lowpan0`. Na figura, é possível observar que a interface `lowpan0` existe e obteve um endereço IPv6 de escopo global.

Para mostrar que esse endereço foi obtido automaticamente, foi feita uma captura de pacotes com Wireshark mostrando os anúncios de rotas feitos pelo *gateway*, que mostram a configuração correta do RADV no mesmo. Essa captura é mostrada na Figura 4.6.

Nessa Figura é possível observar diversos anúncios de rotas feitos pelo *gateway*. O endereço de destino é um endereço de *multicast* e o anúncio do prefixo, que é o importante para a configuração dos endereços, está mostrado em *ICMPv6 Option* no campo *Prefix*, onde se encontra o prefixo de rede obtido pela SIXXS. Assim, como mostrado na Figura 4.5, esse prefixo foi utilizado para montagem do endereço IPv6 de escopo global.

Figura 4.2: RaspberryPi 3 montado como gateway IPv6 e 6LoWPAN [autores]

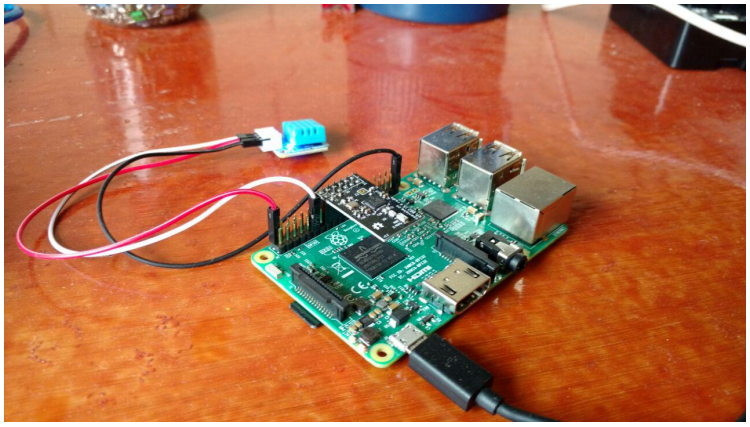


Figura 4.3: RaspberryPi 3 montado como objeto com IPv6 e 6LoWPAN [autores]

Com a rede local configurada, o tráfego de rede em IPv6 deverá ser roteado pelo *gateway*. Para isso, ele precisa do túnel configurado para funcionar com a SIXXS, fazendo o tráfego IPv6 ser encapsulado em datagramas IPv4 para possibilitar tráfego sobre a Internet tradicional. O sucesso da configuração do túnel é mostrado pela saída de dois comandos: primeiro, a verificação do *status* do serviço do AICCU, pois ele é quem fecha o túnel; segundo, a verificação da existência da interface `sixxs` (criada pelo AICCU). Isso é mostrado na Figura 4.7:

O encapsulamento dos datagramas será mostrado nos resultados gerais na seção 4.5.

Uma observação interessante que pode ser feita com a Figura 4.5 é a quantidade de dados trafegados na interface (4,9 MiB recebidos e 1,6 MiB transmitidos). Esses dados possuem mais significado quando leva-se em conta que o objeto está ligado a 10 dias, conforme mostra a saída do comando `uptime` na Figura 4.8, o que dá uma média de tráfego 160 KB por dia (considerando apenas transmissão).

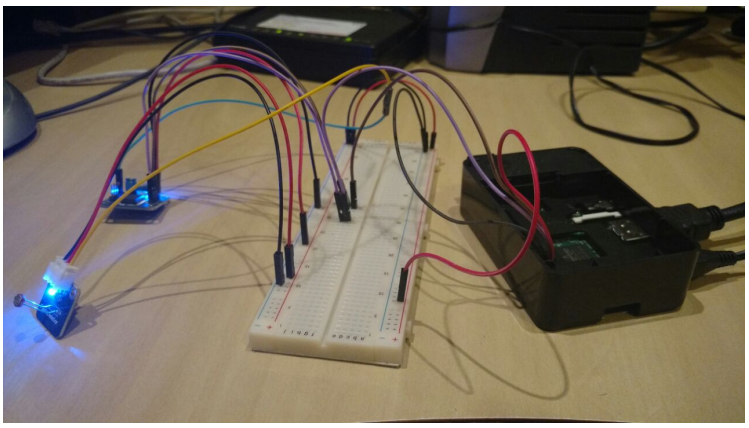


Figura 4.4: RaspberryPi 3 montado como objeto com IPv6 [autores]

```
pi@raspberrypi:~ $ ifconfig lowpan0
lowpan0   Link encap:UNSPEC  HWaddr 18-C0-FF-EE-1A-C0-FF-EE-00-00-00-00-00-00-00-00
          inet6 addr: fe80::1ac0:ffee:1ac0:ffee/64 Scope:Link
          inet6 addr: 2001:1291:200:85b2:1ac0:ffee:1ac0:ffee/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1280  Metric:1
          RX packets:51601 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11805 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:5207864 (4.9 MiB)  TX bytes:1680721 (1.6 MiB)
```

Figura 4.5: Saída do comando `ifconfig lowpan0` no objeto

4.4 FUNCIONAMENTO DA REDE NA CENTRAL

Conforme descrito na seção 3.5, a central é formada por três servidores, sendo um roteador *dual stack*, um servidor Web e cliente CoAP e um servidor de banco de dados (conforme Figura 3.4).

Para mostrar o sucesso da configuração de rede do roteador, é mostrada a saída do comando `ifconfig` na Figura 4.9 com a configuração de rede das duas interfaces. Nessa figura, cada uma da interfaces possui dois endereços de rede, um IPv4 e um IPv6. O único endereço válido IPv4 é o endereço da interface externa (interface `eth0` na `rede_wan`), pois a rede interna (interface `eth1` na`rede_vms`) utiliza um endereço local e NAT para acesso à Internet em IPv4.

Essa verificação foi repetida para os demais servidores. Na Figura 4.10 é mostrada saída do mesmo comando para o `srv-coap` que é o servidor Web e o cliente CoAP. Na Figura 4.11 é mostrada a saída para o `srv-bd`, que é o servidor de banco de dados com MongoDB.

Dessas figuras, infere-se o correto funcionamento da rede para a configuração desejada, com a interface de cada servidor possuindo um endereço IPv4 local e um endereço IPv6 de escopo global. Para mostrar o funcionamento do tráfego de saída em IPv4 e em IPv6, são feitas requisições HTTP simples para dois endereços conhecidos, conforme mostrado na Figura 4.12.

Por fim, a Figura 4.13 mostra os anúncios de rotas para configuração automática das interfaces do servidor de banco de dados e do cliente CoAP (servidor Web). Essa captura é análoga à captura mostrada na Figura 4.6. Os endereços mostrados nas figuras 4.10 e 4.11 foram configurados a partir do prefixo presente nesses anúncios de rotas.

```
No.  Time       Source                Destination           Protocol Length Info
     1 0.000000  fe80::1ac0:ffee:1ac0:ffff ff02::1           ICMPv6       86 Router Advertisement from 18:c0:ff:ee:1a:c0:ff:ff
     2 22.825098 fe80::1ac0:ffee:1ac0:ffff ff02::1           ICMPv6       86 Router Advertisement from 18:c0:ff:ee:1a:c0:ff:ff
     3 41.908365 fe80::1ac0:ffee:1ac0:ffff ff02::1           ICMPv6       86 Router Advertisement from 18:c0:ff:ee:1a:c0:ff:ff

▸ Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
▸ IEEE 802.15.4 Data, Dst: Broadcast, Src: 18:c0:ff:ee:1a:c0:ff:ff
▸ 6LoWPAN
▸ Internet Protocol Version 6, Src: fe80::1ac0:ffee:1ac0:ffff, Dst: ff02::1
▾ Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x8b5a [correct]
    [Checksum Status: Good]
    Cur hop limit: 255
  ▸ Flags: 0x00
    Router lifetime (s): 90
    Reachable time (ms): 0
    Retrans timer (ms): 0
  ▾ ICMPv6 Option (Prefix information : 2001:1291:200:85b2::/64)
      Type: Prefix information (3)
      Length: 4 (32 bytes)
      Prefix Length: 64
    ▸ Flag: 0x60
      Valid Lifetime: 86400
      Preferred Lifetime: 14400
      Reserved
      Prefix: 2001:1291:200:85b2::
  ▸ ICMPv6 Option (Source link-layer address : 18:c0:ff:ee:1a:c0:ff:ff)
```

Figura 4.6: Captura de pacotes de anúncios de rotas feitos pelo gateway

```
pi@raspberrypi:~ $ systemctl status aiccu.service
● aiccu.service - LSB: SixXS Automatic IPv6 Connectivity Client Utility
   Loaded: loaded (/etc/init.d/aiccu)
   Active: active (running) since Thu 2016-10-06 00:27:14 UTC; 1 weeks 3 days ago
  Process: 11824 ExecStop=/etc/init.d/aiccu stop (code=exited, status=0/SUCCESS)
  Process: 11833 ExecStart=/etc/init.d/aiccu start (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/aiccu.service
           └─11885 /usr/sbin/aiccu start
pi@raspberrypi:~ $ ifconfig sixxs
sixxs     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet6 addr: fe80::1091:200:5b2:2/64 Scope:Link
          inet6 addr: 2001:1291:200:5b2::2/64 Scope:Global
          inet6 addr: fe80::3560:848b:bef:1150/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1280  Metric:1
          RX packets:28211 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33685 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:15525298 (14.8 MiB)  TX bytes:17352203 (16.5 MiB)
```

Figura 4.7: Status do serviço AICCU e saída do comando `ifconfig sixxs`

4.5 RESULTADOS GERAIS COM CAPTURA DE PACOTES

Para mostrar o funcionamento geral da prova de conceito, foram realizadas três capturas de pacotes utilizando o comando `tcpdump`. Essas capturas visam obter dados para mostrar a pilha de protocolos completa em três pontos cruciais da rede: a interface 6LoWPAN do *gateway* dos objetos, a interface `sixxs` (túnel) do mesmo *gateway* e a interface do cliente CoAP (também servidor Web), que recebe os dados enviados pelos objetos.

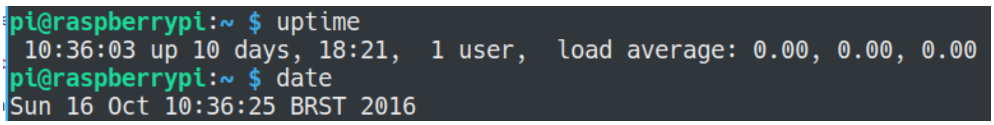


Figura 4.8: Saída do comando *uptime* no objeto

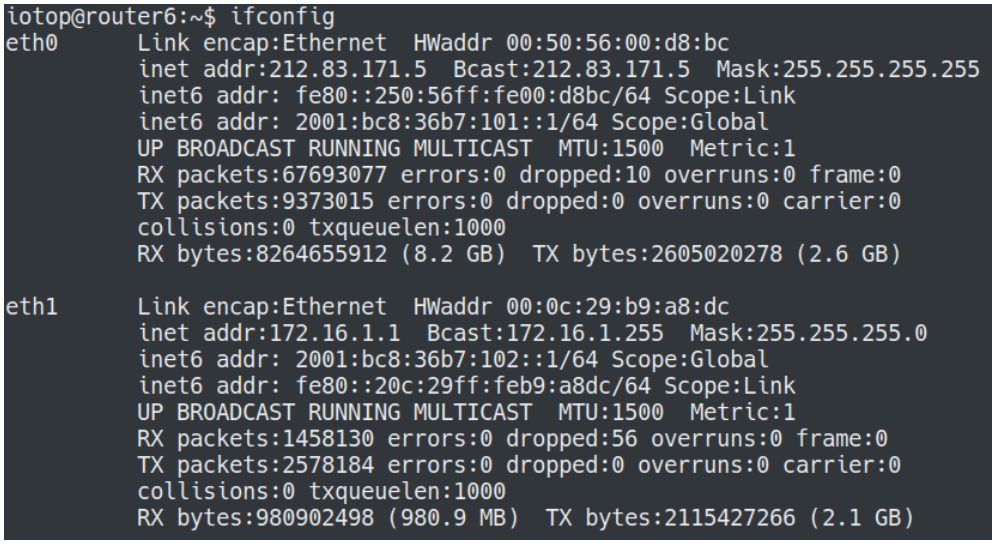


Figura 4.9: Saída do comando `ifconfig` do roteador dual stack na central

A Figura 4.14 mostra um trecho da captura de pacotes feita na interface interna (`lowpan0`) do *gateway*. Nessa Figura, é possível observar toda a pilha de protocolos para IoT, composta por CoAP na camada de aplicação, UDP na camada de transporte, IPv6 com 6LoWPAN na camada de rede e, finalmente, 802.15.4 na camada de enlace. Essa Figura prova o funcionamento de todos os elementos presentes nesse trabalho que envolvem a rede 6LoWPAN dos objetos.

Vale destacar os endereços de origem e destino mostrados na captura. O endereço de origem é o endereço do objeto, conforme mostra a figura 4.5 enquanto o endereço de destino é o endereço IPv6 da única interface do `srv-coap`, conforme mostrado na figura 4.10. Ao expandir os detalhes da figura (Figura 4.15) é possível observar que o MTU do 802.15.4 (127 bytes) não é suficiente para enquadrar inteiramente um datagrama IPv6. Assim, ele é fragmentado, conforme indica o campo *fragmentation header* do cabeçalho. O Wireshark simplifica a visualização e mostra em quantos quadros o datagrama foi transmitido (e qual o tamanho de cada um deles). No caso, foram transmitidos três quadros, com 104, 96 e 17 bytes, respectivamente. Além disso, é possível observar que não houve compressão de cabeçalho IPv6.

Na interface externa do *gateway* (`wlan0`) as capturas (Figura 4.16) mostram o encapsulamento dos datagramas IPv6 dentro de datagramas UDP e então em IPv4, permitindo o tráfego IPv6 através de acesso á Internet residencial IPv4.

```
iotop@srv-coap:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:7c:ae:38
          inet addr:172.16.1.100  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: 2001:bc8:36b7:102:20c:29ff:fe7c:ae38/64 Scope:Global
          inet6 addr: fe80::20c:29ff:fe7c:ae38/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3834069 errors:0 dropped:49 overruns:0 frame:0
          TX packets:3420292 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1479622908 (1.4 GB)  TX bytes:1309445268 (1.3 GB)
```

Figura 4.10: Saída do comando `ifconfig` do servidor Web e cliente CoAP

```
iotop@srv-bd:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:4b:6a:41
          inet addr:172.16.1.101  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4b:6a41/64 Scope:Link
          inet6 addr: 2001:bc8:36b7:102:20c:29ff:fe4b:6a41/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3821085 errors:0 dropped:132 overruns:0 frame:0
          TX packets:2104830 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1425169393 (1.4 GB)  TX bytes:350261516 (350.2 MB)
```

Figura 4.11: Saída do comando `ifconfig` do servidor de banco de dados

Nessa figura, vale observar a presença do protocolo AYIYA, que é um protocolo de encapsulamento de datagramas utilizado pela SIXXS. Normalmente, esse encapsulamento é feito inserindo os datagramas IPv6 dentro de datagramas UDP (que são, então, transportados sobre IPv4) - e é exatamente isso que é mostrado no trecho da captura de pacotes apresentado. Observa-se nos campos IPv6 os endereços de origem e destino referentes respectivamente à interface `lowpan0` do objeto e ao cliente CoAP na central. Além disso, nos campos IPv4, um endereço local de origem (`192.168.0.22`) representa a interface externa (`wlan0`) do objeto enquanto o endereço de destino é o endereço válido que representa o *endpoint* do túnel da SIXXS.

Para provar que esse endereço é da SIXXS, foi feito uma requisição DNS reversa utilizando o comando `nslookup` e o resultado é mostrado na Figura 4.17. Nota-se na figura a saída `name = brudi01.sixxs.net`, provando que o endereço é da SIXXS.

Quando o datagrama chega à SIXXS, ele é desencapsulado e roteado normalmente pela Internet em IPv6 até chegar à central. No momento em que chega na central, o datagrama IPV6 é idêntico ao que saiu do objeto. Há uma modificação somente na camada de enlace, pois a central utiliza Ethernet e não 6LoWPAN. Esse resultado pode ser visualizado na Figura 4.18, o que mostra que o datagrama foi roteado corretamente e os dados chegaram à central.

Essa captura de pacotes mostra a presença do protocolo de camada de aplicação CoAP. Nessa captura, os objetos estão enviando mensagens de resposta a uma relação de `OBSERVE` previamente criada com a central. Essas respostas são mensagens do tipo `NON`, ou seja, não necessitam de confirmação de recebimento (`ACK`) e são enviadas como respostas a cada 5 minutos (por padrão).

```
lotop@srv-coap:~$ wget google.com.br
--2016-10-16 11:24:36--  http://google.com.br/
Resolving google.com.br (google.com.br)... 2a00:1450:4001:81b::2003, 172.217.22.35
Connecting to google.com.br (google.com.br)|2a00:1450:4001:81b::2003|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.google.com.br/ [following]
--2016-10-16 11:24:44--  http://www.google.com.br/
Resolving www.google.com.br (www.google.com.br)... 2a00:1450:4001:81b::2003, 172.217.22.35
Reusing existing connection to [google.com.br]:80.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html'

    [ <=>

2016-10-16 11:24:52 (64.9 MB/s) - 'index.html' saved [10817]
```

Figura 4.12: Saída do comando `wget google.com.br` do servidor de Web

```
No.  Time         Source                Destination           Protocol  Length  Info
  11 25.712375    fe80::20c:29ff:feb9:a8dc ff02::1           ICMPv6       150  Router Advertisement from 00:0c:29:b9:a8:dc
  12 43.481793    fe80::20c:29ff:feb9:a8dc ff02::1           ICMPv6       150  Router Advertisement from 00:0c:29:b9:a8:dc
  13 53.483503    fe80::20c:29ff:feb9:a8dc ff02::1           ICMPv6       150  Router Advertisement from 00:0c:29:b9:a8:dc
▸ Frame 11: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits)
▸ Ethernet II, Src: Vmware_b9:a8:dc (00:0c:29:b9:a8:dc), Dst: IPv6mcast_01 (33:33:00:00:00:01)
▸ Internet Protocol Version 6, Src: fe80::20c:29ff:feb9:a8dc, Dst: ff02::1
▾ Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0xd7c5 [correct]
    [Checksum Status: Good]
    Cur hop limit: 64
  ▸ Flags: 0x00
    Router lifetime (s): 90
    Reachable time (ms): 0
    Retrans timer (ms): 0
  ▾ ICMPv6 Option (Prefix information : 2001:bc8:36b7:102::/64)
      Type: Prefix information (3)
      Length: 4 (32 bytes)
      Prefix Length: 64
    ▸ Flag: 0xc0
      Valid Lifetime: 86400
      Preferred Lifetime: 14400
      Reserved
      Prefix: 2001:bc8:36b7:102::
```

Figura 4.13: Captura de tela mostrando os anúncios de rota na central

Essas mensagens carregam um objeto representado em notação JSON. Cada *resource* responde com um objeto diferente contendo o valor lido, o identificador do objeto que enviou o dado, o nome do *resource* e o carimbo de tempo (*timestamp*) do momento em que a medida foi obtida, o que é fundamental para gerar os gráficos mostrados posteriormente aos usuários. A Figura 4.19 mostra uma captura de tela contendo uma mensagem CoAP do tipo NON com esse *payload* sendo carregado do objeto para a central.

Logo após os dados chegarem à central, o servidor de aplicação (que é o mesmo que o cliente CoAP) abre uma conexão com o servidor de banco de dados para gravar os dados recebidos. Por isso, na mesma captura de pacotes é possível visualizar essa comunicação usando a porta TCP 27017 (apelidada de `MONGO`), conforme mostra a Figura 4.20.

O dado recebido pela central é um objeto na notação JSON. Esse objeto é gravado sem alterações como um documento na coleção correspodnente do MongoDB.

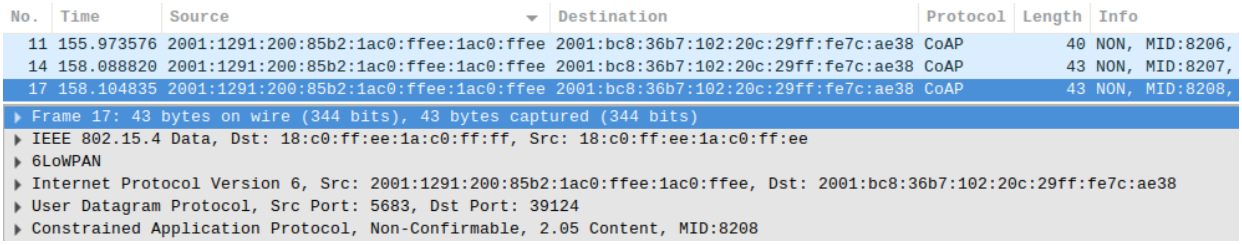


Figura 4.14: Exibição da captura de pacotes na interface `lowpan0` do gateway

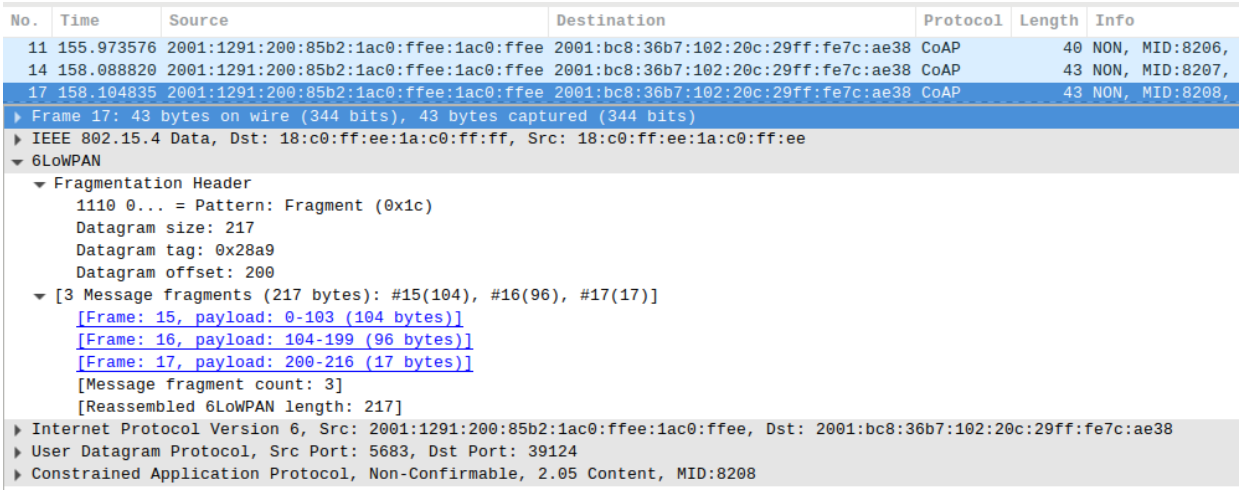


Figura 4.15: Exibição da captura de pacotes na interface `lowpan0` do gateway. Detalhes da fragmentação do datagrama IPv6

4.6 APLICAÇÃO EM FUNCIONAMENTO

Uma vez que os dados são salvos no banco de dados, a aplicação é utilizada para mostrar os mesmos aos usuários, além de realizar algumas operações básicas de controle com os objetos (como determinar que deve ser criada uma relação de `OBSERVE` com um dado *resource*).

O controle é feito na tela principal da aplicação, em que são listados os objetos previamente cadastrados e os *resources* disponíveis para cada um deles mostrados na Figura 4.21. Além disso, para demonstração, foi incluído um mapa que mostra a localização dos objetos, do *endpoint* da SIXXS e do local na França onde está o servidor contratado (central). A Figura 4.22 mostra o mapa geral mostrando os marcadores para cada um desses elementos.

Como os objetos estão muito próximos no mapa, foi capturada uma nova imagem com mais proximidade, mostrando os dois objetos utilizados e o *endpoint* do túnel da SIXXS. Esse pedaço do mapa pode ser visto na Figura 4.23.

Na Figura 4.24, é possível visualizar Brasília, mostrando as diferentes localizações dos dois objetos utilizados no trabalho.

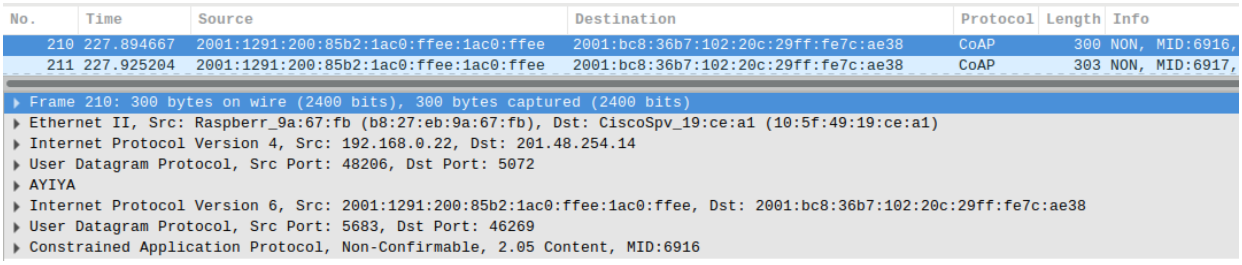


Figura 4.16: Exibição da captura de pacotes na interface `wlan0` (externa) do gateway

```
[jg@jg-note ~]$ nslookup 201.48.254.14
14.254.48.201.in-addr.arpa      name = brudi01.sixxs.net.
```

Figura 4.17: Resultado da requisição DNS reversa para o endereço IP válido da SIXXS

A segunda página Web da aplicação é a página de gráficos. Para montar essa página, a aplicação solicita ao banco de dados uma agregação que mostra todos os objetos que possuem ou possuíram gráficos salvos em algum momento. Essa tela permite que o usuário selecione qual objeto terá os gráficos exibidos. Essa parte é apresentada na Figura 4.25.

Após clicar em alguma opção da figura 4.25, os gráficos para todos os *resources* ativos serão exibidos, conforme mostra a Figura 4.26. Essa tela permite que o usuário filtre qualquer intervalo de dados entre a última hora e as últimas 24 horas através de um *slider* interativo, conforme os exemplos mostrados nas Figuras 4.26, 4.27, 4.29 e 4.28 , em que estão selecionados dois intervalos diferentes: 24h (padrão) e 2h, para os *resources* de temperatura, umidade e intensidade de luz.

Os gráficos das Figuras 4.26 e 4.28 estão coerentes com o que se espera de gráficos de temperatura e umidade. Primeiro, a temperatura diminui de madrugada (por volta de 1h da manhã), alcança seu valor mínimo entre 6h e 8h da manhã e retorna a subir. Existe uma relação entre temperatura e umidade, em que um aumento de temperatura normalmente está relacionado à uma diminuição na umidade e vice-versa. Isso mostra que os sensores estão funcionando, embora não sejam precisos.

Os gráficos das Figuras 4.27 e 4.29 mostram a intensidade da luz em um quarto residencial no intervalo de 24h e 2h respectivamente. Com o intervalo de 24h é possível perceber a intensidade de luz diminuindo perto de 23h e depois se mantendo constante (esse valor não diminuiu mais devido ao led do sensor que mostra o funcionamento do mesmo). A intensidade de luz volta a aumentar perto de 7h da manhã quando o sol começa a nascer.

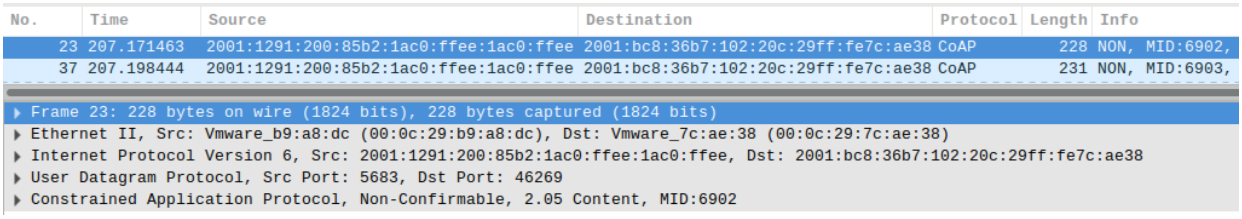


Figura 4.18: Trecho da captura de pacotes mostrando os dados chegando à central

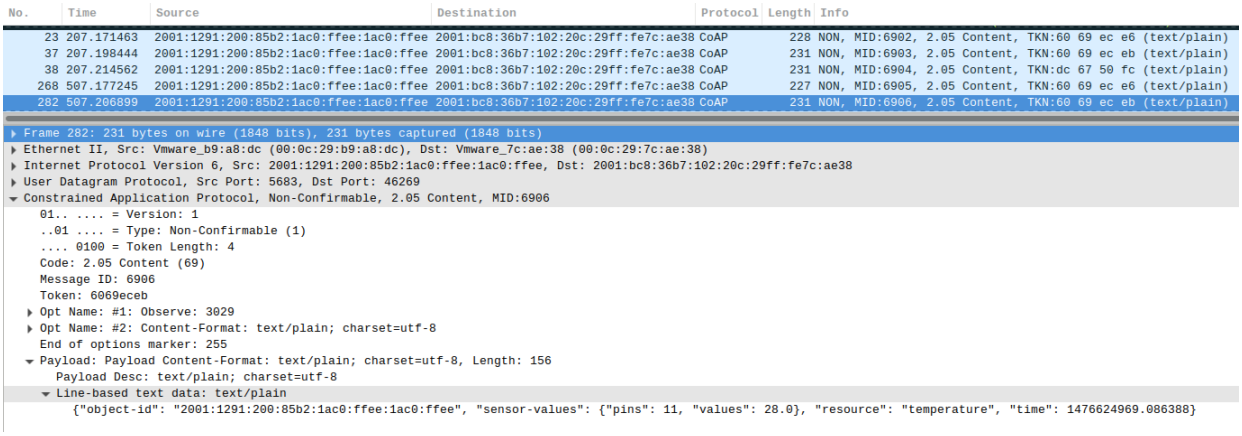


Figura 4.19: Trecho da captura de pacotes mostrando os dados chegando à central - detalhe no objeto JSON recebido

4.6.1 Bot do Telegram

Para checar o último valor registrado no banco de dados com maior facilidade, pode ser utilizado o Bot para o mensageiro Telegram. Esse bot é capaz de identificar o usuário pelo seu `telegram id` e assim filtrar os dados referentes ao objeto de determinado usuário. As capturas de tela tiradas do aplicativo em execução no sistema operacional Android mostram a inicialização e o funcionamento do bot após um pedido de *lastValue*, que solicita o último valor gravado no banco de dados. Essas capturas são mostradas nas Figuras 4.30 e 4.31.

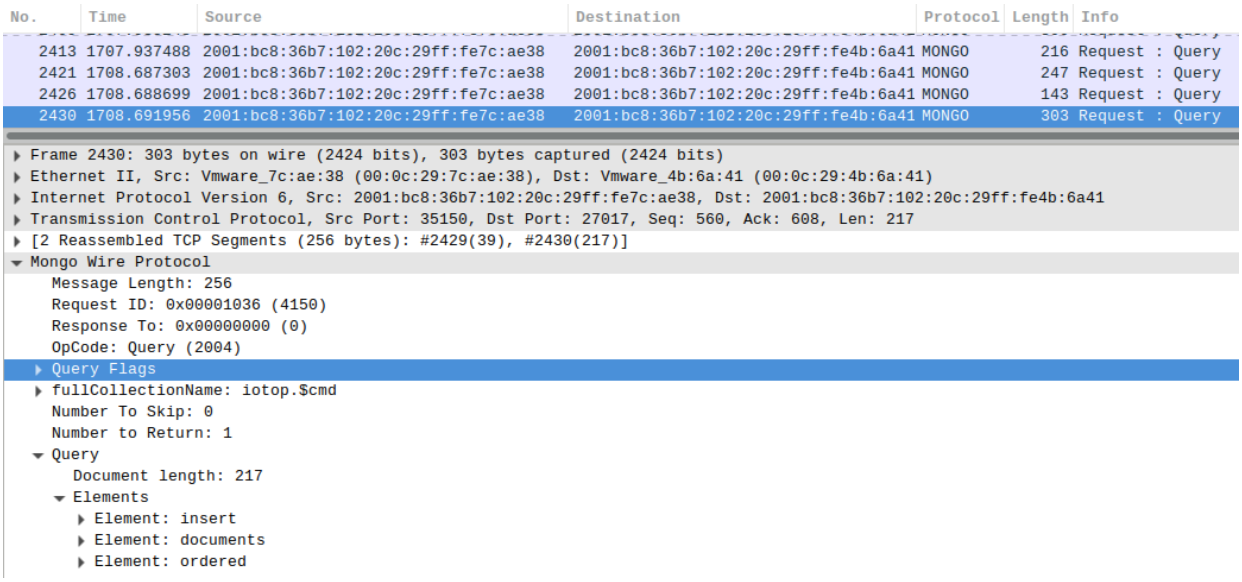


Figura 4.20: Trecho da captura de pacotes mostrando os dados sendo enviados ao banco de dados

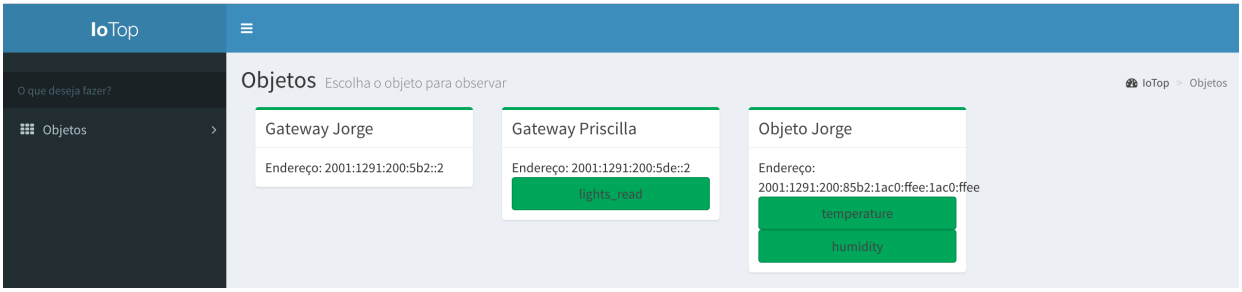


Figura 4.21: Lista de objetos com seus respectivos resources

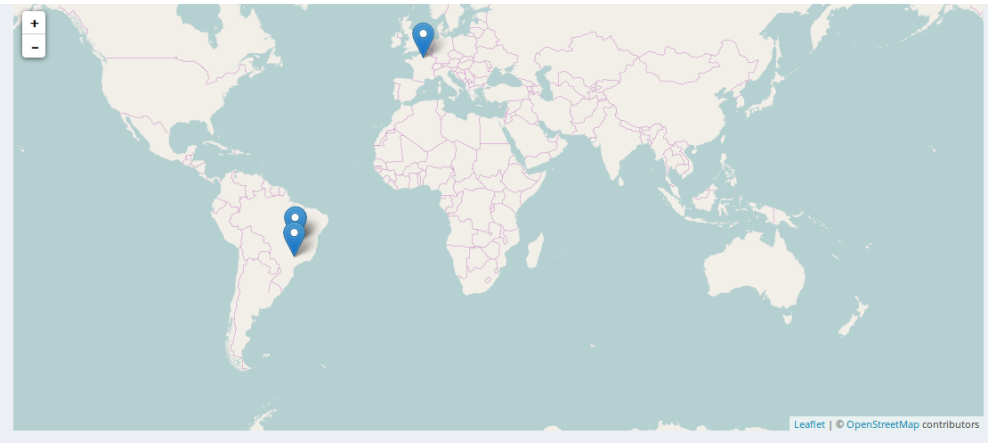


Figura 4.22: Mapa geral mostrando todos os pontos

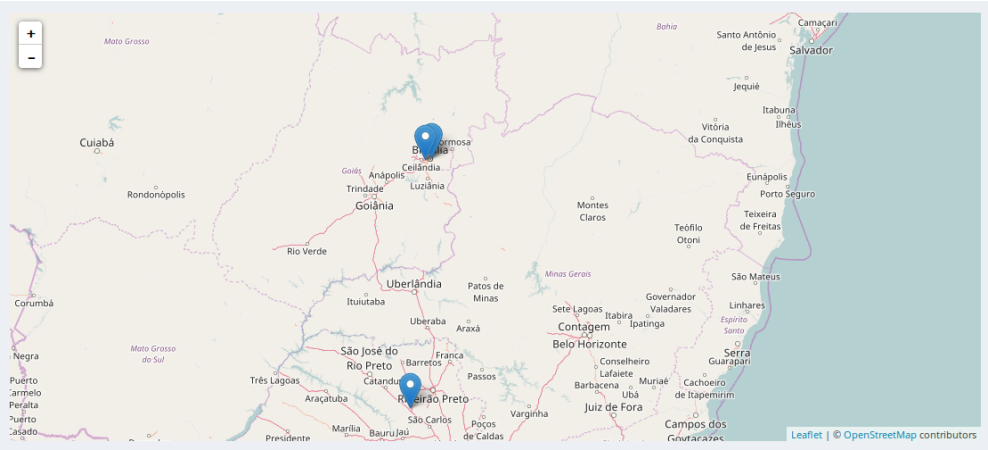


Figura 4.23: Mapa geral mostrando os pontos com aproximação visual no Brasil

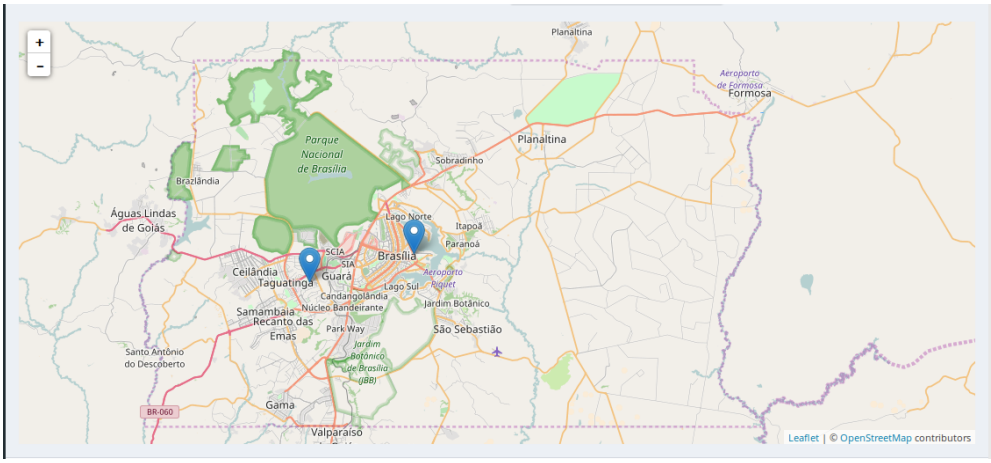


Figura 4.24: Mapa geral mostrando os pontos com aproximação visual no Brasília

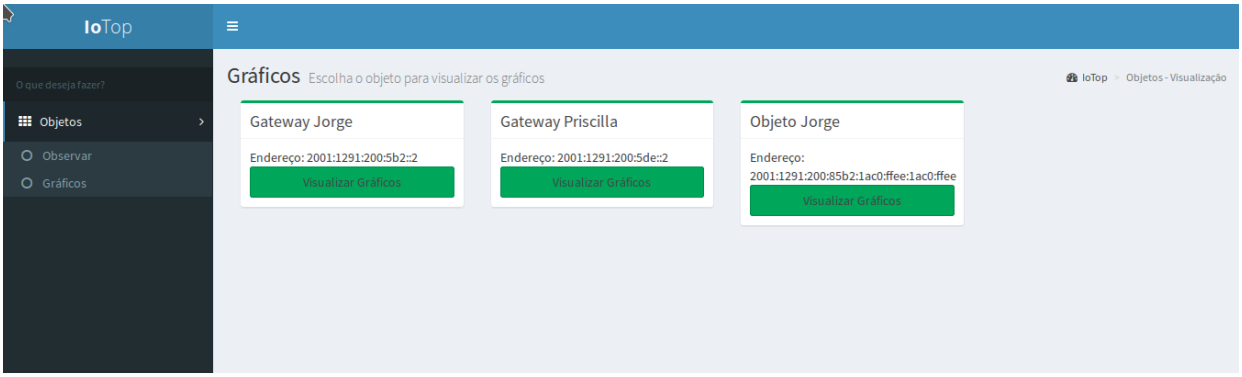


Figura 4.25: Tela de seleção de objeto para exibição dos gráficos

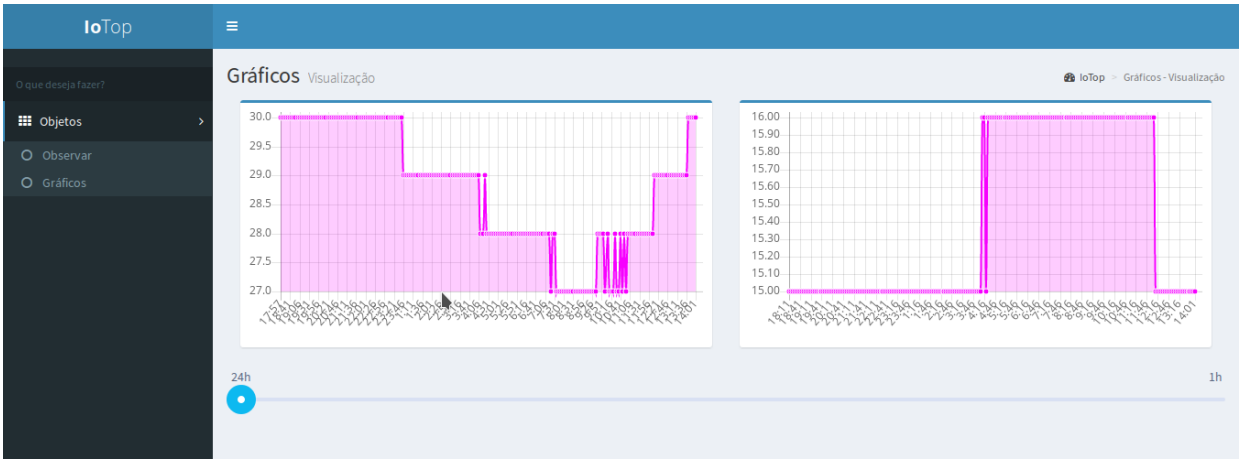


Figura 4.26: Gráficos de temperatura e umidade em intervalo das últimas 24h

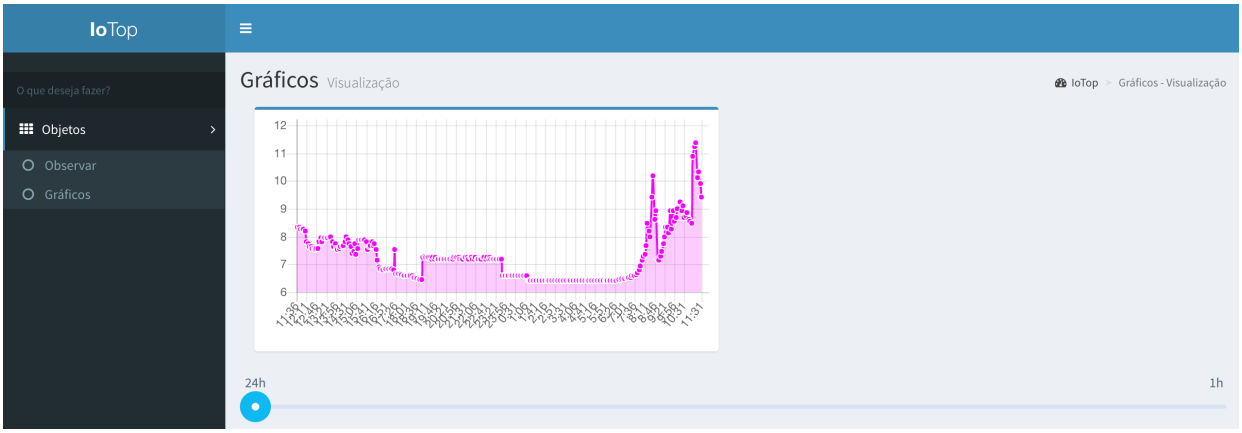


Figura 4.27: Gráficos de intensidade de luz em intervalo das últimas 24h

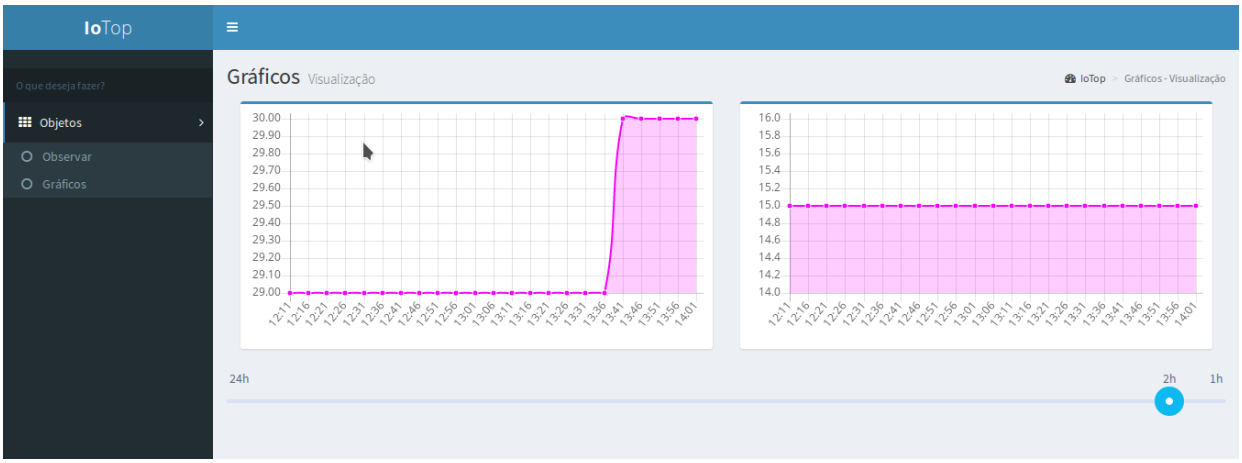


Figura 4.28: Gráficos de temperatura e umidade em intervalo das últimas 2h

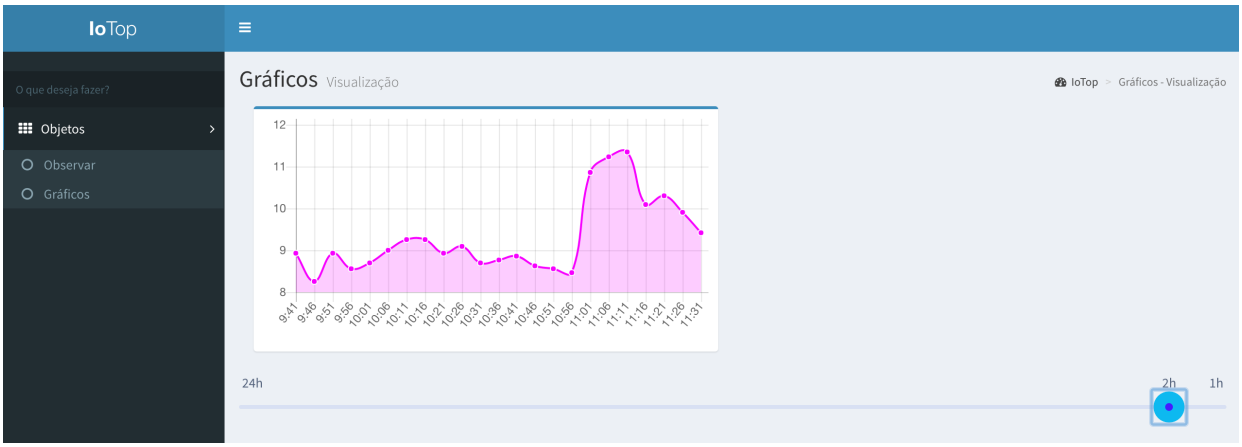


Figura 4.29: Gráficos de intensidade de luz em intervalo das últimas 2h

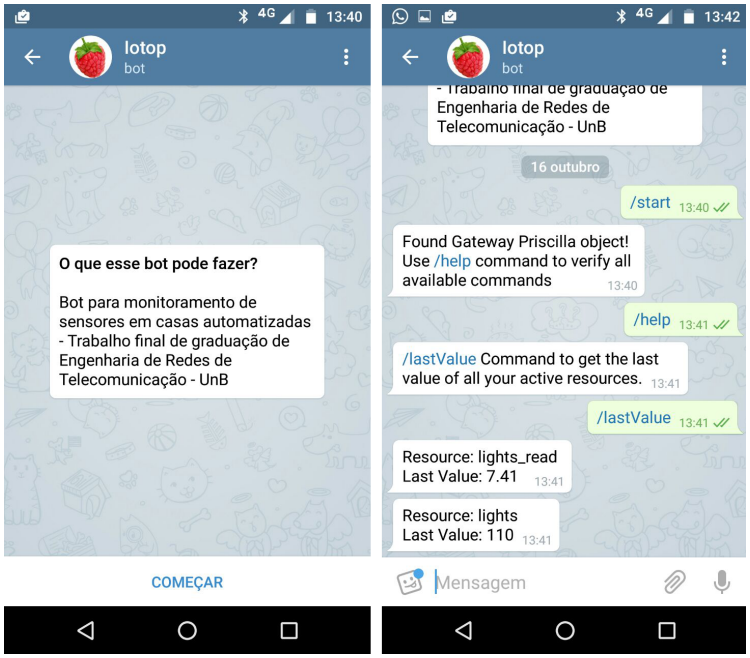


Figura 4.30: Bot do Telegram em execução para o objeto do primeiro usuário

Figura 4.31: Bot do Telegram em execução para o objeto do segundo usuário

5 CONCLUSÃO E TRABALHOS FUTUROS

Dados os resultados apresentados no capítulo 4, a rede proposta funciona como planejado, cumprindo os objetivos de ser uma prova de conceito, ou seja, demonstrar o funcionamento da rede em uma implementação real.

Foi possível visualizar dois objetos se comunicando com uma central, sendo um deles utilizando o Wi-Fi na camada de enlace e o outro utilizando IEEE 802.15.4, o que permitiu a utilização do 6LoWPAN como camada de adaptação entre enlace e rede. Além disso, os resultados mostram a nova pilha de protocolos sugerida em diversos artigos para a implementação apropriada de IoT em dispositivos de baixo consumo de energia. Essa pilha mostrada como funcional é composta por IPv6, UDP e CoAP nas camadas de rede, transporte e aplicação respectivamente.

O protocolo CoAP merece atenção especial, pois ele que permitiu a utilização dos objetos como servidores de aplicação e não como clientes, como normalmente é feito. Isso retira uma carga de processamento da central, que passa a não precisar realizar *polling* nos objetos para obter os dados. Além disso, como foi utilizado o servidor de aplicação Actinium, é possível realizar a substituição do código dos objetos remotamente pela central, o que fica como sugestão para trabalhos futuros. A utilização do próprio Actinium pode precisar ser revista com o tempo, pois a especificação do protocolo CoAP (RFC 7252) ainda não está concluída (trata-se de um *proposed standard*) e por isso tende a sofrer modificações com o tempo, como a recente adição da RFC 7959 finalizada em Agosto de 2016.

Ficou claro, também, a utilização de IPv6 em uma residência que não possui endereçamento IPv6 nativo, o que foi possível graças aos serviços gratuitos de tunelamento da empresa SIXXS. Contudo, devido à lenta adoção do protocolo IPV6 na Internet, a SIXXS se pronunciou recentemente negando a prestação do serviço para novos usuários como forma de incentivar a implementação de IPv6 de forma nativa pelos provedores de acesso. Além disso, o uso de túneis é uma solução temporária que não precisará ser levada em consideração quando a Internet utilizar majoritariamente IPv6.

Na camada de enlace, por sua vez, foi utilizado o 6LoWPAN pela sua baixa taxa de transferência e baixo consumo de energia, útil para os dispositivos simples tipicamente usados com sensores na IoT. Contudo, devido às implementações *opensource* dos módulos para Linux, não há ainda a implementação de todas as funcionalidades, como por exemplo a muito necessária compressão do cabeçalho IPv6. Implementar o 6LoWPAN de maneira definitiva com utilização de compressão de cabeçalho também fica como sugestão para trabalhos futuros - isso ajudaria a evitar a fragmentação em nível de enlace.

Há também uma implementação sugerida para roteamento na rede, uma vez que a rede apresentada é estática com topologia estrela, ou seja, todos os objetos da rede são capazes de se comunicar utilizando diretamente o *gateway*. Em cenaŕios reais, os objetos não tem potência de transmissão suficiente para isso e, considerando um ambiente com múltiplos objetos, seria interessante utilizar roteamento *multi-hop* em que diversos saltos (através de outros objetos) são realizados até que um dado objeto possa alcançar o *gateway*.

Todas as séries questões de segurança não foram abordadas no trabalho e é fortemente sugerido a implementação de mecanismos de segurança como trabalhos futuros. Como exemplo, o próprio Actinium (servidor de aplicação CoAP) possui pronta uma implementação DTLS (TLS com UDP) para funcionar com o *framework* Californium. Seria possível combinar as funcionalidades utilizadas nesse trabalho com o DTLS para se obter uma solução um pouco mais segura. Além disso, é necessário tratar a criptografia dos dados em toda a comunicação fim-a-fim (todas as camadas) incluindo a própria central com o seu banco de dados.

Com a rede existente também podem ser realizados testes de desempenho para avaliar como a aplicação Web funciona em um ambiente com muitos usuários e também com muitos objetos, o que é o esperado da IoT. Pode-se simular a presença de milhões de objetos enviando dados para uma mesma central e então observar como o sistema se comporta do ponto de vista do desempenho e da estabilidade. Essa possibilidade também fica como sugestão de trabalho futuro.

Por fim, a prova de conceito apresentada utiliza apenas a leitura de dados de sensores. Esses dados são mais úteis quando o ambiente se torna mais interativo e inteligente, quando os dados recolhidos são utilizados para atuar diretamente em outros objetos, como motores, lãmpadas, aparelhos domésticos e computadores, por exemplo. O fato da rede proposta utilizar os objetos como servidores e esses possuírem endereços de escopo global facilita a comunicação direta com os objetos. Essa ideia também fica como sugestão de trabalho futuro.

REFERÊNCIAS BIBLIOGRÁFICAS

Al-Fuqaha et al. 2014 AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things for smart cities. *IEEE Internet of Things Journal*, v. 1, p. 22–32, 2014.

Al-Fuqaha et al. 2015 AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, p. 2347–2376, 2015.

Atzori, Iera e Morabito 2010 ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, p. 2787–2805, 2010.

Canonical 2016 CANONICAL. *Ubuntu*. 2016. Disponível em: <http://www.ubuntu.com/>. Acesso em: 05 de Junho de 2016.

CEPTRO-BR 2016 CEPTRO-BR. *Porcentagem de utilização de endereços IPv6 no Brasil*. 2016. Disponível em: <http://rosabaya.ceptro.br/ipv6-measurement-charts/>. Acesso em: 31 de Maio de 2016.

Debian 2016 DEBIAN. *Sobre o Debian*. 2016. Disponível em: <https://www.debian.org/intro/about>. Acesso em: 05 de Junho de 2016.

Distrowatch 2016 DISTROWATCH. *Page Hit Rank*. 2016. Disponível em: <http://distrowatch.com/dwres.php?resource=popularity>. Acesso em: 05 de Junho de 2016.

Eclipse 2014 ECLIPSE. *Californium (Cf) CoAP*. 2014. Disponível em: <http://www.eclipse.org/californium/>. Acesso em: 08 de Outubro de 2016.

HUInfinito 2016 HUINFINITO. *HU Infinito Componentes Eletrônicos*. 2016. Disponível em: <http://www.huinfinito.com.br/>. Acesso em: 05 de Outubro de 2016.

IEEE IEEE. *Ethernet*. Disponível em: <http://www.networksorcery.com/enp/protocol/IEEE8023.htm>. Acesso em: 05 de Junho de 2016.

IEEE 2016 IEEE. *IEEE 802.11 Wireless Local Area Networks – The Working Group for WLAN Standards*. 2016. Disponível em: <http://www.ieee802.org/11/>. Acesso em: 06 de Outubro de 2016.

IETF 1980 IETF. *RFC 768 – User Datagram Protocol*. 1980. Disponível em: <https://www.ietf.org/rfc/rfc768.txt>. Acesso em: 01 de Junho de 2016.

IETF 1981 IETF. *RFC 793 – Transmission Control Protocol*. 1981. Disponível em: <https://tools.ietf.org/html/rfc793>. Acesso em: 01 de Junho de 2016.

IETF 1997 IETF. *Dynamic Host Configuration Protocol*. 1997. Disponível em: <https://www.ietf.org/rfc/rfc2131>. Acesso em: 05 de Junho de 2016.

IETF 1997 IETF. *Dynamic Host Configuration Protocol*. 1997. Disponível em: <https://www.ietf.org/rfc/rfc3315>. Acesso em: 05 de Junho de 2016.

IETF 1998 IETF. *RFC 2460 – Internet Protocol, Version 6*. 1998. Disponível em: <https://tools.ietf.org/html/rfc2460>. Acesso em: 31 de Maio de 2016.

IETF 2001 IETF. *RFC 3053 – IPv6 Tunnel Broker*. 2001. Disponível em: <https://tools.ietf.org/html/rfc3053>. Acesso em: 04 de Junho de 2016.

IETF 2005 IETF. *RFC 4213 – Basic IPv6 Transition Mechanisms*. 2005. Disponível em: <https://tools.ietf.org/html/rfc4213>. Acesso em: 04 de Junho de 2016.

IETF 2007 IETF. *RFC 4861 – Neighbor Discovery for IP version 6 (IPv6)*. 2007. Disponível em: <https://tools.ietf.org/html/rfc4861>. Acesso em: 31 de Maio de 2016.

IETF 2007 IETF. *RFC 4862 – IPv6 Stateless Address Autoconfiguration*. 2007. Disponível em: <https://tools.ietf.org/html/rfc4862>. Acesso em: 01 de Junho de 2016.

IETF 2007 IETF. *RFC 4941 – Privacy Extensions for Stateless Address Autoconfiguration in IPv6*. 2007. Disponível em: <https://tools.ietf.org/html/rfc4941>. Acesso em: 03 de Novembro de 2016.

IETF 2010 IETF. *RFC 5942 – IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes*. 2010. Disponível em: <https://tools.ietf.org/html/rfc5942>. Acesso em: 03 de Novembro de 2016.

IETF 2014 IETF. *RFC 7252 – The Constrained Application Protocol (CoAP)*. 2014. Disponível em: <https://tools.ietf.org/html/rfc7252>. Acesso em: 05 de Junho de 2016.

IETF 2015 IETF. *RFC 7641 – Observing Resources in the Constrained Application Protocol (CoAP)*. 2015. Disponível em: <https://tools.ietf.org/html/rfc7641>. Acesso em: 05 de Junho de 2016.

IETF 2016 IETF. *Hypertext Markup Language - 2.0*. 2016. Disponível em: <https://tools.ietf.org/html/rfc1866>.

IETF 2016 IETF. *Hypertext Transfer Protocol – HTTP/1.1*. 2016. Disponível em: <https://www.ietf.org/rfc/rfc2616.txt>.

IETF 2016 IETF. *Opportunistic Security for HTTP*. 2016. Disponível em: <http://www.ietf.org/id/draft-ietf-httpbis-http2-encryption-05.txt>. Acesso em: 09 de Junho de 2016.

IETF 2016 IETF. *RFC 2373 – IP Version 6 Addressing Architecture*. 2016. Disponível em: <https://www.ietf.org/rfc/rfc2373.txt>. Acesso em: 03 de Outubro de 2016.

IETF 2016 IETF. *RFC 6177 – IPv6 Address Assignment to End Sites*. 2016. Disponível em: <https://tools.ietf.org/html/rfc6177>. Acesso em: 03 de Outubro de 2016.

IPv6BR 2012 IPV6BR. *IPv6.br - Transição*. 2012. Disponível em: <http://ipv6.br/post/transicao/>. Acesso em: 05 de Junho de 2016.

Kurose e Ross 2012 KUROSE, J.; ROSS, K. *Computer Networking: A Top Down Approach*. [S.l.]: Addison-Wesley, 2012.

LACNIC 2016 LACNIC. *Contador de endereços IPv4 restantes*. 2016. Disponível em: <http://www.lacnic.net/pt/web/lacnic/inicio>. Acesso em: 31 de Maio de 2016.

MEAN 2016 MEAN. *MEAN Stack*. 2016. Disponível em: <http://mean.io/#!/>. Acesso em: 05 de Outubro de 2016.

MongoDB 2016 MONGODB. *MongoDB Manual*. 2016. Disponível em: <https://docs.mongodb.com/manual/core/databases-and-collections/>. Acesso em: 05 de Junho de 2016.

Morimoto 2009 MORIMOTO, C. E. *Linux: guia prático*. Porto Alegre, RS, Brasil: Sul Editores, 2009.

Mozilla 2016 MOZILLA. *JavaScript - Language Guide*. 2016. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>. Acesso em: 05 de Outubro de 2016.

NIC.br NIC.BR. *IPv6 nas redes de sensores*. Disponível em: <http://www.ceptro.br/pub/CEPTRO/PalestrasPublicacoes/FISL10-6LoWPAN-v6.pdf>. Acesso em: 05 de Junho de 2016.

NIC.br 2012 NIC.BR. *Ative e use o IPv6*. 2012. Disponível em: <http://ipv6.br/post/ative-e-use-o-ipv6/>. Acesso em: 04 de Junho de 2016.

NodeJS 2016 NODEJS. *NodeJS*. 2016. Disponível em: <https://nodejs.org/en/>. Acesso em: 05 de Outubro de 2016.

Oracle 2012 ORACLE. *Java - Language Guide*. 2012. Disponível em: <http://docs.oracle.com/javase/8/docs/technotes/guides/language/>. Acesso em: 05 de Outubro de 2016.

Python 2016 PYTHON. *The Python Tutorial*. 2016. Disponível em: <https://docs.python.org/3/tutorial/index.html>. Acesso em: 05 de Junho de 2016.

RaspberryPi 2016 RASPBERRYPI. *Raspbian*. 2016. Disponível em: <https://www.raspberrypi.org/downloads/raspbian/>. Acesso em: 05 de Junho de 2016.

RIOTMakers 2016 RIOTMAKERS. *Setup native 6LoWPAN router using Raspbian and RADVD*. 2016. Disponível em: <https://github.com/RIOT-Makers/wpan-raspbian/wiki/Setup-native-6LoWPAN-router-using-Raspbian-and-RADVD>. Acesso em: 03 de Outubro de 2016.

RIOTMakers 2016 RIOTMAKERS. *Spice up Raspbian for the IoT – Systemd lowpan*. 2016. Disponível em: <https://github.com/RIOT-Makers/wpan-raspbian/wiki/Spice-up-Raspbian-for-the-IoT#systemd-lowpan>. Acesso em: 03 de Outubro de 2016.

SixXS 2015 SIXXS. *AICCU – Automatic IPv6 Connectivity Client Utility*. 2015. Disponível em: <https://www.sixxs.net/tools/aiccu/>. Acesso em: 06 de Junho de 2016.

SixXS 2015 SIXXS. *Setting Up an IPv6 home network with Ubuntu*. 2015. Disponível em: <https://www.sixxs.net/wiki/Setting_Up_an_IPv6_home_network_with_Ubuntu>. Acesso em: 06 de Junho de 2016.

SixXS 2016 SIXXS. *Pontos de Presença*. 2016. Disponível em: <https://www.sixxs.net/main/>. Acesso em: 01 de Dezembro de 2016.

SixXS 2016 SIXXS. *Pontos de Presença*. 2016. Disponível em: <https://www.sixxs.net/pops/>. Acesso em: 04 de Junho de 2016.

Sunfounder SUNFOUNDER. *Sensor Kit 2.0 for Raspberry Pi B+*. Disponível em: <https://www.sunfounder.com/learn/category/sensor-kit-v2-0-for-raspberry-pi-b-plus.html>. Acesso em: 16 de Outubro de 2016.

Tan e Koo 2014 TAN, J.; KOO, S. A survey of technologies in internet of things. *IEEE International Conference on Distributed Computing in Sensor Systems*, p. 269–274, 2014.

W3School W3SCHOOL. *HTML - Introduction*. Disponível em: <http://www.w3schools.com/html/html_intro.asp>. Acesso em: 07 de Junho de 2016.

APÊNDICES

I. CONFIGURAÇÕES E CÓDIGOS RELEVANTES

I.1 CÓDIGOS DE LEITURA DE SENSORES

I.1.1 Temperatura e Umidade (DHT11)

Trecho de código (em python) que utiliza a biblioteca da Adafruit para ler dados de temperatura e umidade do sensor DHT11:

```
#!/usr/bin/env python

# Adafruit_DHT instalada com pip
# RPi.GPIO acompanha Raspbian
import Adafruit_DHT as DHT
import RPi.GPIO as GPIO

# [...]

def read_temperature(pin):
    # pino 17 fixo por possuir tensao de 3.3 V
    return DHT.read_retry(pin, 17)

# [...]

# Le dados
def loop():
    while True:
        # [...]
        # configs e um dicionario no python contendo as configuracoes descritas
        # em um arquivo externo
        humidity, temperature = read_temperature(configs["sensors"]["pins"]["temperature"])

        # leitura de dados...
        # [...]
        # Aguarda intervalo de tempo
        time.sleep(10)
```

Nesse código, a função `read_temperature` recebe o número do pino em que o sensor DHT11 está ligado. Esse valor não está fixo no código, mas sim é passado através de um arquivo configuração que é lido em outro momento (aqui omitido) e é representado por:

```
configs["sensors"]["pins"]["temperature"]
```

Após isso, o código continua (trecho novamente omitido) e então tudo é repetido após o intervalo de 10 segundos utilizando a função `time.sleep`.

I.1.2 Fotoresistor através de um conversor analógico digital

Conforme apresentado na seção 2.5.1.3, o fotoresistor necessita do conversor analógico-digital para converter o seu sinal de dados. Como explicado da seção 2.5.1.2 esse sensor necessita do envio de um endereço em hexadecimal para que ele realize a conversão correta. O endereço 0x48 em hexadecimal (e em binário 01001000) foi utilizado nesse projeto. Para realizar a leitura dos pinos, foi necessário inicializar os pinos corretos. Com essa parte configurada, basta fazer a leitura do pino que contêm a saída do conversor. Segue abaixo os trechos:

```
#!/usr/bin/env python
import PCF8591 as ADC
import RPi.GPIO as GPIO
import time
import yaml
import netifaces
import json

def setup():
    ADC.setup(0x48)
    GPIO.setup(DO, GPIO.IN)
    stream = open("config.yaml", "r")
        global configs
        configs = yaml.load(stream)

[...]

def loop():
    status = 1
    while True:
        print 'Real Value:', ADC.read(0)
        a = float(1000)/ADC.read(0)
        b = float("{0:.2f}".format(a))
        print 'Value: ', b
        time.sleep(0.2)
```

I.1.3 Integração com o Actinium

O trecho de código abaixo é um pedaço do código em Python que executa nos objetos fazendo a leitura dos dados dos sensores. Os dados dos sensores são então gravados em arquivos para serem lidos e transmitidos na rede pelo servidor de aplicação Actinium.

Esse trecho mostra a montagem do dicionário Python (chamado de `sensor_data`) a partir dos dados lidos pela função `read_temperature` apresentada anteriormente e a gravação em arquivo JSON do objeto correspondente a esse dicionário:

```
##### TEMPERATURA #####
# Cria objeto no formato de representacao de objetos do Python
sensor_data = {
    "object-id": get_if_global_6address(configs["interface"]),
    "time": time.time(),
    "resource": "temperature",
    "sensor-values": {
        "pins": (11),
        "values": (temperature)
    },
}

# Gera o objeto JSON e o grava em arquivo
write_sensors_file(configs["files"]["sensors_out"]["temperature"], sensor_data)

###### UMIDADE ######
# Cria objeto no formato de representacao de objetos do Python
sensor_data = {
    "object-id": get_if_global_6address(configs["interface"]),
    "time": time.time(),
    "resource": "humidity",
    "sensor-values": {
        "pins": (11),
        "values": (humidity)
    },
}

# Gera o objeto JSON e o grava em arquivo
write_sensors_file(configs["files"]["sensors_out"]["humidity"], sensor_data)
```

A função `write_sensors_file` citada acima é a função responsável por criar os arquivos (caso não existam) ou substituir o conteúdo dos arquivos antigos. É ela que efetivamente faz a conversão dos dicionários Python em objetos JSON utilizando a função `json.dump` (proveniente da biblioteca externa `json`):

```
# Gravacao do arquivo .json com dados dos sensores
def write_sensors_file(file_name, sensor_data):
    # Cria arquivo caso nao exista
    out_file = open(file_name, "w+")
    out_file.close()

    # Primeiro limpa conteudo do arquivo
    with open(file_name, "r+") as out_file:
        out_file.seek(0)
        out_file.truncate()

    # em seguida grava conteudo novo
    out_file = open(file_name, "w+")
    json.dump(sensor_data, out_file)
    out_file.close()
```

Com esse código, são gerados três arquivos JSON, um para cada *resource*. Para expandir a leitura dos dados para mais sensores, basta adicionar suas medidas em um dicionário semelhante aos que foram apresentados acima e salvar um novo arquivo. O arquivo de destino deverá ser especificado no arquivo de configuração com o nome do *resource*, conforme explicado na seção 3.6.1.

I.2 CÓDIGO DO ACTINIUM

O trecho abaixo apresenta o trecho do código em Actinium que trata requisições do tipo `GET` que chegam ao objeto:

```
app.root.onget = function (exchange) {

    var bufferedReader =
        new java.io.BufferedReader(
            new java.io.FileReader("/home/pi/sensors/temperature.json"));
    var strCurrentLine = "";

    strCurrentLine = bufferedReader.readLine();

    exchange.respond( 2.05, strCurrentLine );

}
```

O código apresentado é um código em Javascript. Porém, graças ao uso do interpretador Rhino, é possível acessar algumas funções de objetos de bibliotecas nativas da API Java, como `java.io`. No trecho apresentado, o sistema abre o conteúdo do arquivo JSON gerado pelo código em Python através do uso de um `FileReader` em conjunto com um `BufferedReader` - ambos objetos nativos da API Java (`java.io`). Então, como os arquivos possuem apenas uma linha de texto, o método `readLine` do `BufferedReader` é chamado apenas uma vez e o conteúdo dessa linha é enviado como resposta à central através do método `exchange.respond` - nativo do Actinium.

A função de temporização (chamada de `funcaoContadora`) é apresentada abaixo. Ela utiliza o método `sleep` nativo da API Java (`java.lang`), fazendo com que a `thread` que executa o código fique pausada por 5 minutos:

```
function funcaoContadora() {

    while (true) {

        java.lang.Thread.sleep(temporizador);
        app.root.changed();
    }

}
```

Após o tempo de 5 minutos, a *thread* continua e a sinalização para envio da resposta à central é feita com o método `changed`. Esse método pertence à `app.root`, que representa, em Javascript, a raíz do *resource* (no caso, `temperature`) em execução em um *app* no Actinium (cada código em execução é um *app* em execução no mesmo). Com isso, o Actinium se encarrega de enviar a linha de texto lida para a central.

Para que o método `changed` funcione, é preciso notificar ao Actinium que esse *resource* é "observável", isto é, passível de criação de relações de `OBSERVE`, conforme estabelecido na RFC 7641 [IETF 2015]. Isso é feito através da linha abaixo, colocada ao final do código:

```
app.root.setObservable(true);
```

I.3 CONFIGURAÇÃO PROXY REVERSO HTTP COM CACHE E SSL

O NGINX foi utilizado como proxy reverso atendendo a requisições HTTP e também como terminador de conexões SSL, para conexão segura com os usuários. O trecho relevante do arquivo principal de configuração é apresentado abaixo:

```
http {
     # [...]
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request"'
                      '"$status $body_bytes_sent "$http_referer"'
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
    proxy_cache_path /var/cache/nginx keys_zone=cache01:10m;
    include /etc/nginx/conf.d/*.conf;
}
```

O trecho acima apresenta um pedaço da seção `http`, que claramente se refere à função de servidor Web do NGINX. Primeiramente, é definido um formato para os arquivos de log pela diretiva `log_format`. Esse formato é chamado de `main` e é utilizado pela diretiva `access_logs`, que especifica que o arquivo de log que salva os logs dos acessos deverá possuir entradas nesse formato e será salvo no arquivo especificado (`access.log`).

Na sequência, é definido o local onde é feito o cache de conteúdo estático, seu respectivo identificador e o tamanho de memória compartilhada usada para armazenar informações de metadados sobre arquivos em cache (que, no caso, é de 10 MB).

Por fim, é especificada a diretiva `include`. Nesse caso, ela especifica que deverão ser carregados todos os arquivos de configuração terminados em `.conf` dentro do diretório `conf.d`. Isso permite que as configurações sejam modularizadas, o que é útil em um ambiente de hospedagem de múltiplas páginas, pois é possível manter a configuração de cada página em um arquivo separado, facilitando a gerência de configuração.

Nesse diretório existe apenas um arquivo de configuração para a única página servida através do proxy. Os trechos relevantes da configuração são apresentados a seguir.

```
server {
    server_name     iotop.jorgeguilherme.eng.br;
    access_log      /var/log/nginx/access_iotop.log main;
    error_log       /var/log/nginx/error_iotop.log warn;
    listen          443;
    proxy_cache     cache01;

    ssl on;
    ssl_certificate /etc/ssl/certs/iotop.crt;
    ssl_certificate_key /etc/ssl/private/iotop.key;
    ssl_protocols TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!MD5;

    location / {
            proxy_bind      172.16.1.1;
            proxy_pass      http://172.16.1.100/;
    }

    location /230500968:AAGCF4io0mQhxOMaDH7G4bgHMUiKAWM1xlZ {
        proxy_bind      172.16.1.1;
        proxy_pass      http://172.16.1.100:8443/230500968:AAGCF4io0mQhxOMaDH7G4bgHMUiKAWM1xlZ;
    }
}
```

Esse bloco de configuração (`server`) é usado para especifiar a configuração de um *Virtual Host* específico. Os *Virtual Hosts* permitem que um mesmo servidor Web atenda a requisições provenientes de diferentes domínios. No caso, a configuração nesse bloco se refere exclusivamente ao domínio *iotop.jorgeguilherme.eng.br*, conforme especificado no parâmetro `server_name`. Caso houvesse a necessidade de atender a requisições HTTP de outros domínios, um outro bloco `server` deveria existir (de preferência em um arquivo separado). Essa separação é possível graças ao parâmetro `host` presente no cabeçalho das mensagens HTTP [IETF 2016], pois esse campo é utilizado pelo servidor HTTP para diferenciar requisições encaminhadas a diferentes *Virtual Hosts*.

Nesse bloco, são especificados parâmetros de arquivos de logs diferentes dos parâmetros já definidos anteriormente - isso foi feito para que os logs referentes aos acessos (e erros) encaminhados para esse *Virtual Host* sejam separados dos logs gerais e de outros *Virtual Hosts*, caso esses existissem. A diretiva `proxy_cache` é usada com o nome `cache01`, o mesmo que foi configurado no primeiro arquivo de configuração, o que faz com que os arquivos de cache sejam salvos no diretório especificado anteriormente e que as informações de metadados se limitem aos 10 MB também já especificados.

O parâmetro `listen` especifica que o servidor irá escutar por conexões HTTP na porta 443, utilizada por padrão para HTTPS (HTTP com SSL ou TLS). De fato, o servidor utiliza HTTPS para assegurar a integridade das informações trafegadas e, por isso, se sucede a configuração dos parâmetros SSL. Esses parâmetros são listados abaixo:

- `ssl on`: Habilita o uso de SSL ou TLS;
- `ssl_certificate` e `ssl_certificate_key` especificam o local onde está armazenado o certificado mostrado aos usuários e a chave privada correspondente;
- `ssl_protocols` especifica quais protocolos podem ser utilizados - no caso, foi escolhida a última versão do TLS (1.2). Como exemplos, poderiam ser inseridos também TLS 1.0 e SSL 3.0, de acordo com as necessidades de compatibilidade;
- `ssl_ciphers` especifica quais cifras podem ser (ou não) utilizadas. Essa configuração segue o formato de cifras do OpenSSL.

Então, finalmente são especificas as diretivas de configuração `location`, dentro das quais há dois parâmetros que se referem ao uso do NGINX como proxy reverso:

- `proxy_bind` Diz ao NGINX que ele deve repassar as requisições aos servidores de *upstream* utilizando o endereço IP especificado. Essa configuração é útil para quando o servidor possui múltiplas interfaces de rede, como é o caso.
- `proxy_pass` Diz ao NGINX para quais endereços devem ser encaminhadas as requisições feitas para a URI especificada no parâmetro `location`. No caso, como esse parâmetro especifica a raíz do servidor (`/`), isso significa que todas as requisições feitas para esse *Virtual Host* devem ser encaminhadas para `172.16.1.100`, que é o servidor Web. Como exemplo, se chegar uma requisição ao servidor Proxy com destino a *http://server_name/subdir*, essa requisição será encaminhada para *http://172.16.1.100/subdir*.

Há uma outra configuração `location`, especificando uma porta diferente no servidor de `upstream`. Isso foi feito pois a aplicação no servidor executa um segundo servidor Web utilizada para *webhooks* com uma API externa, conforme será detalhado na seção 3.8.4.

Vale observar que até o momento a configuração apresentada somente atende a solicitações HTTPS na porta 443. Para redirecionar o acesso de conexões HTTP simples, foi criado o trecho de configuração abaixo:

```
server {

    server_name iotop.jorgeguilherme.eng.br;
    listen 80;
    return 301 https://$host$request_uri;
}
```

Esse trecho apresenta a configuração para o mesmo *Virtual Host*, porém, com o servidor escutando na porta 80 (HTTP simples). Tudo que essa configuração determina é que o servidor responda com a mensagem HTTP 301 (*movido permanentemente*), o que indica que o navegador do usuário deverá se conectar com o mesmo servidor, mas utilizando o protocolo HTTPS. Dessa forma, o acesso é garantido ser feito sempre em HTTPS e não em HTTP.

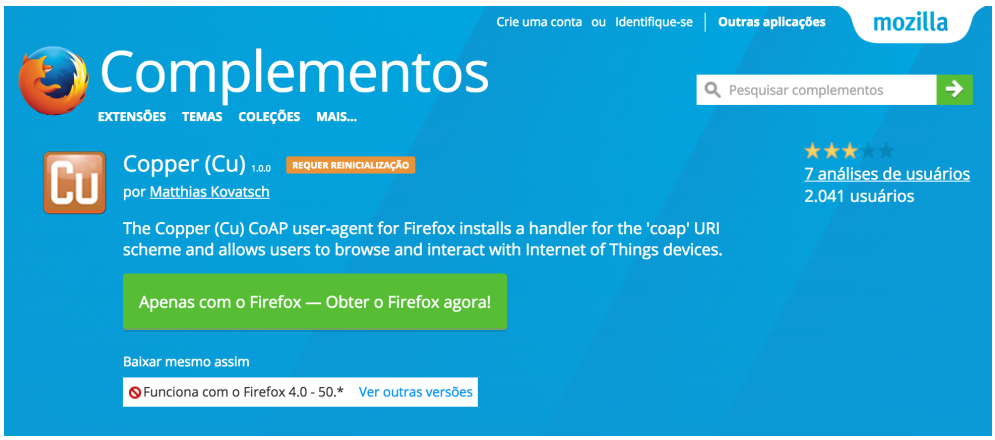


Figura I.1: Site para download do complemento Cupper no *browser* Firefox.

I.4 TELEGRAM - EXEMPLO DE CÓDIGO PARA O BOT

Segue abaixo o trecho de código que trata o comando "help"como exemplo para demonstrar a programação com a API de bots:

```
bot.onText(/\/help/, function(msg){
    var fromId = msg.from.id;
    var resp = '/lastValue Command to get the last value of all your active resources.'
    bot.sendMessage(fromId, resp);
});
```

Esse código responde ao usuário quando o mesmo solicita ajuda para o bot no aplicativo.

Esse comando informa ao usuário a existência de outro comando disponível, "lastValue", que retorna o último valor medido pelo sensor buscando essas informacões e filtrando a partir do *telegramID* do usuário.

I.5 UPLOAD DE CÓDIGOS NO ACTINIUM

Com a aplicação Actinium devidamente instalada no objeto o mesmo funciona como um servidor CoAP. Esse servidor pode ser configurado localmente com a ajuda de um complemento de browser chamado Cupper. Esse instalador está disponível para o *browser* Firefox e fará com que esse *browser* possa se comunicar por meio do protocolo CoAP.

O *download* do complemento pode ser realizado no site <*https://addons.mozilla.org/pt-br/firefox/addon/copper-270430/*>. A Figura I.1 mostra a interface do site.

Com o complemento instalado, agora é possível acessar o servidor CoAP que está rodando no objeto por meio do endereço IPv4 do Raspberry Pi 3. A Figura I.2 mostra a interface de configuração do servidor CoAP do objeto pelo *browser*. Lembre-se de tomar o cuidado de especificar o protocolo (*coap://*) e a porta que o servidor está ouvindo, 5683.

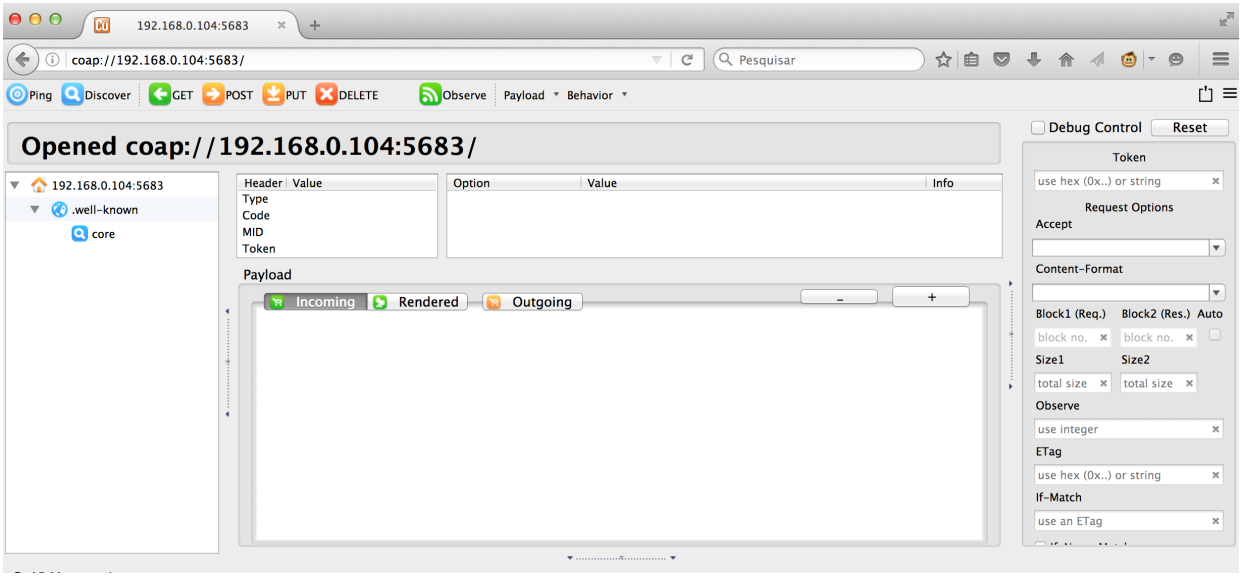


Figura I.2: Interface de configuração do servidor CoAP por meio do *browser* Firefox

Nessa interface é importante salientar o botão superior esquerdo, *Discovery* que implementa o método para coletar todos os *resourcers* instalados no objeto, conforme RFC. A Figura I.3 mostra o que foi descoberto no objeto, focando nas aplicações que já se encontram instaladas no servidor.

Para realizar a instalação de um novo aplicativo, ou seja, fazer o *upload* de um código para o servidor CoAP é necessário enviar uma requisição do método POST com o parâmetro de URL *install?appname* em que *appname* será o nome da sua aplicação. Na Figura I.4 é possível ver destacada a URL que será enviada e o código que deverá ser escrito na seção *Outgoing* do *payload*. Ao realizar o `POST` a instalação é feita com sucesso e o aplicativo aparece na seção de aplicações instaladas *install*, como é destacado na Figura I.5.

Com a aplicação instalada o próximo passo é criar uma instância dessa aplicação no servidor CoAP. Esse passo também é realizado com uma requisição `POST` e uma URL com o parâmetro *appname*. Nesse `POST` é necessário mandar o payload com o seguinte comando: `name = app_teste` em que `app_teste` será o nome da instância. As Figuras I.6 e I.7 mostram a sequência de passos e o resultado final com a instância `app_teste` criada dentro de *Apps* na seção *instances*.

Finalmente, para iniciar a instância no servidor é necessário enviar uma última requisição `POST` com o payload *start* para essa instância. Dessa forma, a aplicação será iniciada no servidor CoAP como mostra e destaca a Figura I.8.

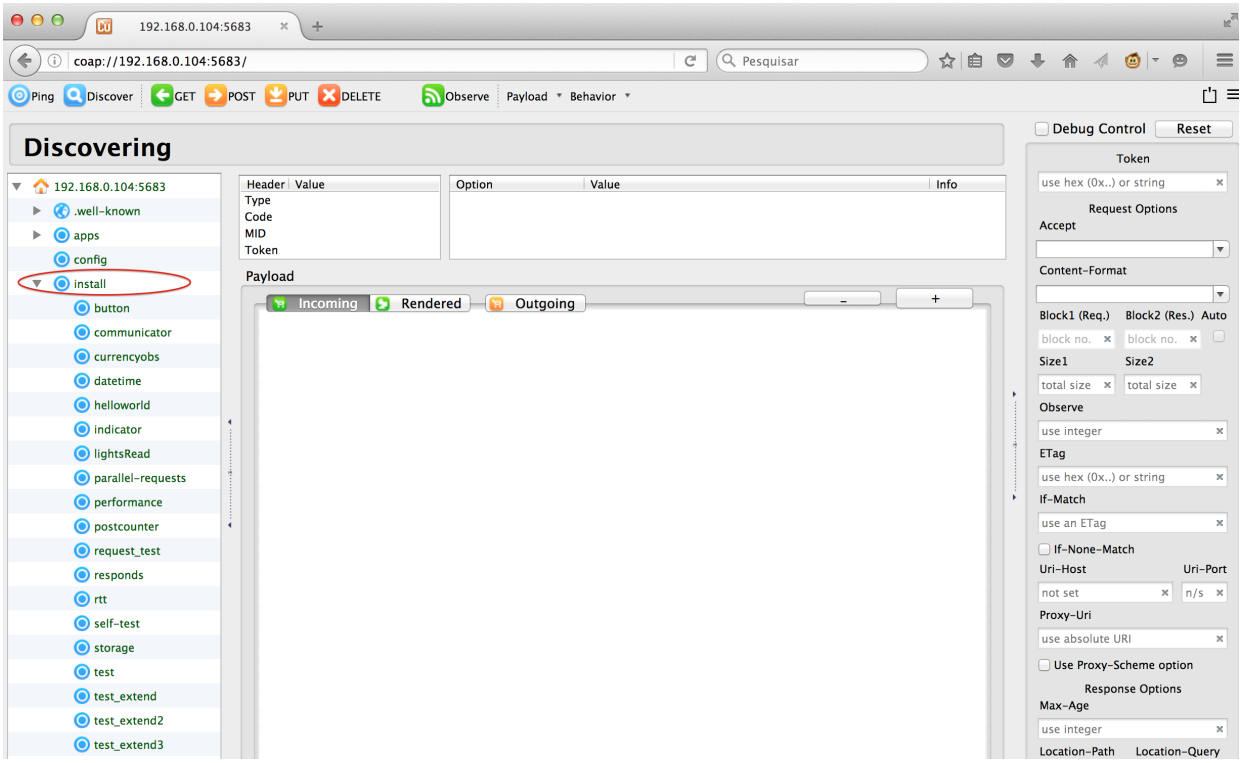


Figura I.3: Aplicações e *resourcers* descobertos pelo método *Discovery*

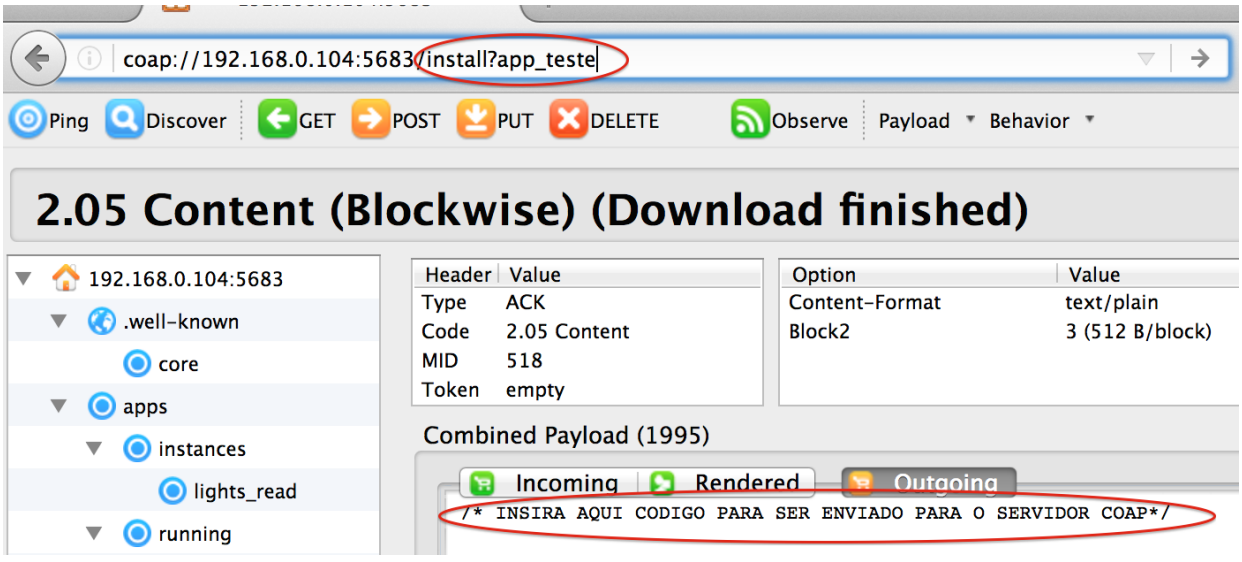


Figura I.4: *Upload* da aplicação app_teste no servidor CoAP

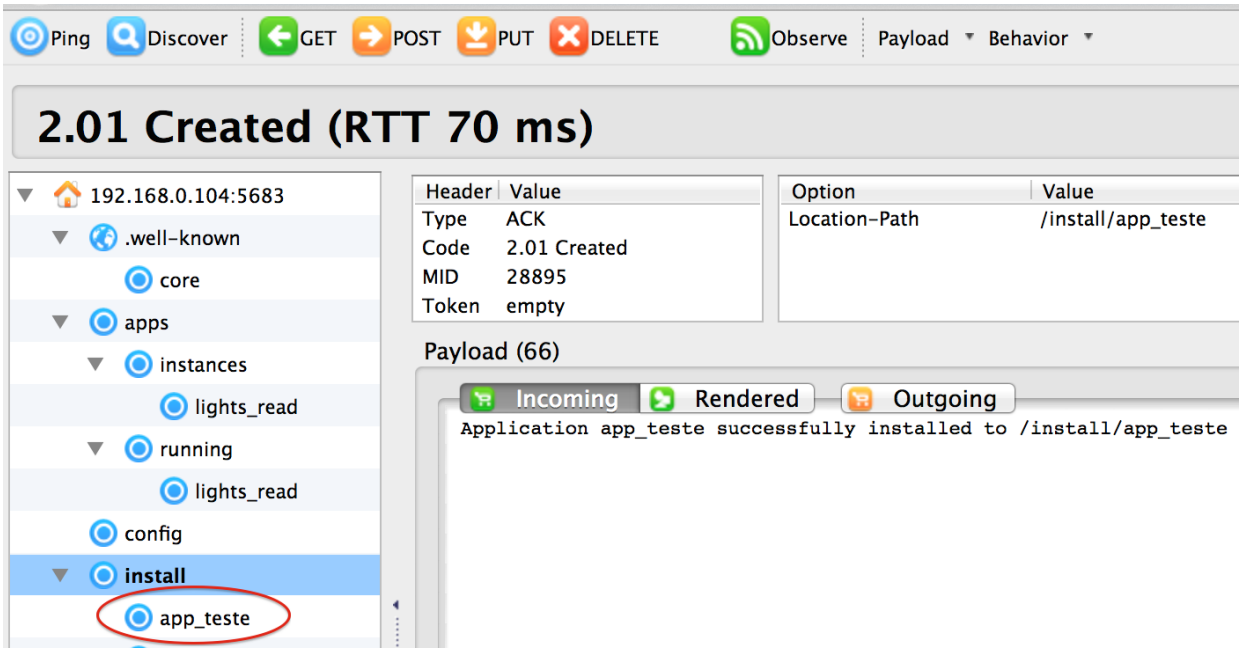


Figura I.5: *Upload* da aplicação app_teste na seção *install*

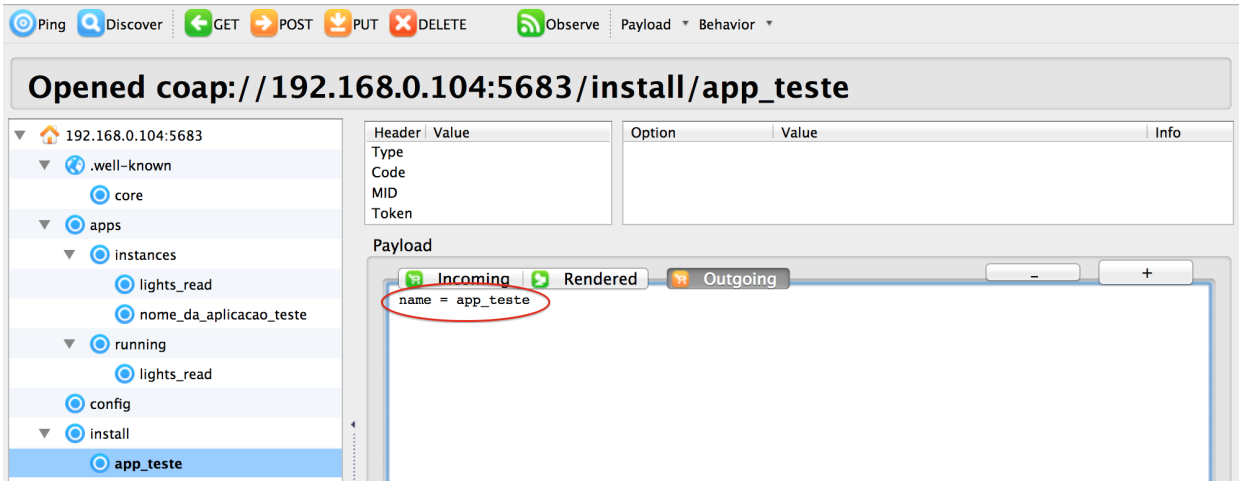


Figura I.6: Criação da instância da aplicação app_teste no servidor.

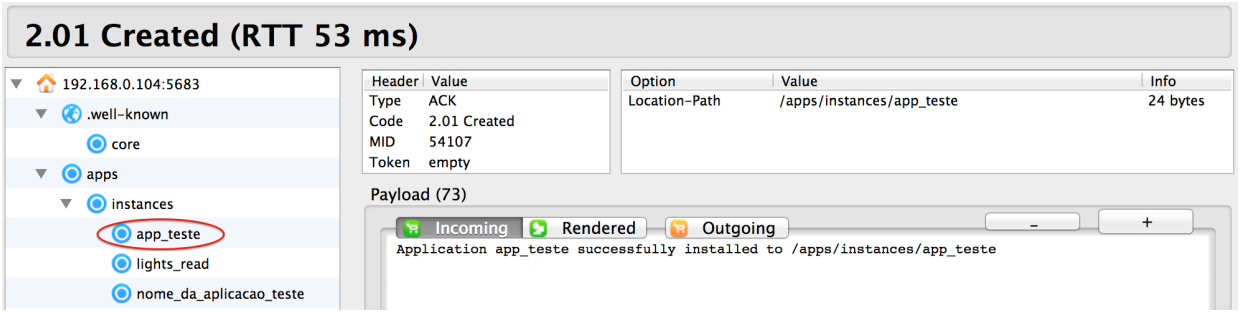


Figura I.7: Instância app_teste criada na seção *instances*

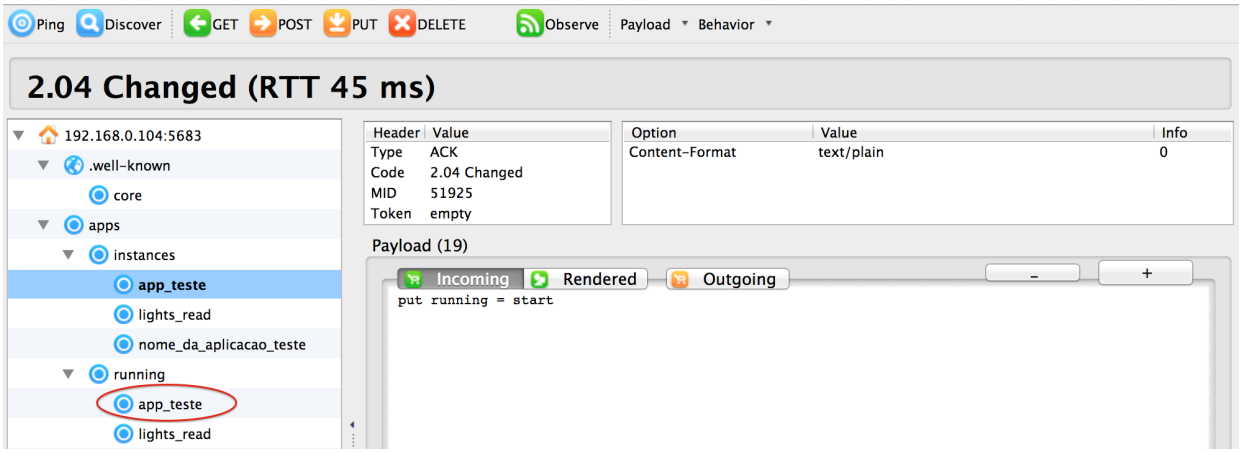


Figura I.8: Aplicação app_teste iniciada no servidor CoAP